

# vFabric GemFire User's Guide

VMware vFabric GemFire 7.0

Note: Documentation for Pivotal GemFire 7.0.x is now available at <http://gemfire.docs.pivotal.io/index.html>. Please refer to the Pivotal site for the latest and most up-to-date documentation on GemFire. The vFabric GemFire documentation site will no longer be updated.

If you have comments about this documentation, submit your feedback to:  
[docfeedback@vmware.com](mailto:docfeedback@vmware.com)

Copyright © 2013 VMware, Inc. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws. VMware products are covered by one or more patents listed at

VMware is a registered trademark or trademark of VMware, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies.

VMware, Inc.  
3401 Hillview Ave.  
Palo Alto, CA 94304

---

# Contents

<b>About vFabric GemFire User's Guide.....</b>	<b>17</b>
<b>Documentation Roadmap.....</b>	<b>19</b>
<b>Part I: Supported Configurations and System Requirements.....</b>	<b>1</b>
<b>Chapter 1: vFabric GemFire Supported Configurations .....</b>	<b>3</b>
Host Machine Requirements.....	4
Dependencies on Linux RPM Packages.....	5
Running vFabric GemFire in Pure Java Mode.....	5
<b>Chapter 2: vFabric GemFire Tools Supported Configurations.....</b>	<b>7</b>
GemFire Pulse System Requirements.....	7
DataBrowser System Requirements.....	7
VSD System Requirements.....	7
<b>Chapter 3: vFabric GemFire Native Client Supported Configurations...9</b>	
<b>Chapter 4: vFabric GemFire Modules Supported Configurations.....13</b>	
<b>Part II: Getting Started with vFabric GemFire.....15</b>	
<b>Chapter 5: vFabric GemFire in 15 Minutes or Less.....17</b>	
<b>Chapter 6: About vFabric GemFire.....21</b>	
<b>Chapter 7: Main Features of vFabric GemFire.....23</b>	
<b>Chapter 8: Installing vFabric GemFire.....27</b>	
Pre-Installation Note for vFabric Suite Customers.....	27
RHEL: Install vFabric GemFire (Standalone) from RPM.....	27
RHEL: Install vFabric GemFire (vFabric Suite) from the VMware YUM Repository.....	30
Install vFabric GemFire from a ZIP File.....	32
How to Uninstall GemFire.....	34
<b>Chapter 9: Upgrading vFabric GemFire.....35</b>	
Before You Upgrade.....	35
RHEL: Upgrade vFabric GemFire (Standalone) from RPM.....	36
RHEL: Upgrade vFabric GemFire (vFabric Suite) from the VMware YUM Repository.....	38
Upgrade vFabric GemFire from a ZIP File.....	41
Performing a Rolling Upgrade.....	43

---

vFabric GemFire Version Compatibility Rules.....	.44
<b>Chapter 10: Understanding vFabric GemFire Licenses.....</b>	<b>45</b>
vFabric Suite and vFabric GemFire Licenses.....	45
vFabric GemFire License Options.....	46
Choosing a License Option Based on Topology.....	46
Checking License Quantity.....	52
How GemFire Manages Licenses.....	52
<b>Chapter 11: Installing and Configuring vFabric GemFire Licenses....</b>	<b>55</b>
Option 1: Install Serial Numbers in gemfire.properties.....	55
Option 2: Install Serial Numbers in vFabric Serial Number Files.....	57
vFabric Suite Only: Configure GemFire for vFabric License Server.....	58
Troubleshooting Issues with vFabric GemFire Licenses.....	59
<b>Chapter 12: Setting Up the Product Examples.....</b>	<b>61</b>
<b>Chapter 13: vFabric GemFire Tutorial.....</b>	<b>65</b>
Tutorial Introduction.....	65
vFabric GemFire Features Demonstrated in the Tutorial.....	65
Sample Social Networking Application.....	66
Running the Tutorial.....	66
Exploring Other Features with the Tutorial .....	78
<b>Chapter 14: Hello World Example.....</b>	<b>81</b>
Introduction to the Hello World Example.....	81
Walkthrough of the Example Files.....	82
<b>Chapter 15: QuickStart Examples.....</b>	<b>85</b>
Setting Up Your Environment.....	85
Management and Monitoring Example.....	85
JSON Example.....	86
Replicated Caching Example.....	88
Server Managed Caching Example.....	89
Partitioned Data Caching Example.....	90
Fixed Partitioning.....	92
Reliable Event Notification Examples.....	93
Persisting Data to Disk Example.....	97
Overflowing Data to Disk Example.....	98
Cache Expiration Example.....	99
Cache Eviction Example.....	100
Querying Example (OQL).....	101
Transactions Example.....	102
Delta Propagation Example.....	102
Function Execution Example.....	104

Secure Client Example.....	105
Multiple Secure Client Example.....	106
Distributed Locking Example.....	107
Benchmark Examples.....	108
Troubleshooting Checklist.....	110

## **Part III: Getting Started with Spring Data GemFire .....113**

### **Chapter 16: About vFabric GemFire.....115**

### **Chapter 17: About Spring Data GemFire.....117**

Overview of Application Development with Spring Data GemFire.....	117
Recommended Development Model.....	118
Spring Data GemFire Code Examples.....	119
Additional Resources.....	119

### **Chapter 18: Configuring GemFire Components with the Spring Framework.121**

System Requirements.....	121
Downloading vFabric GemFire and Spring Data GemFire Components.....	121
Configuring Caches.....	122
Configuring Regions.....	123
Configuring Cache Listeners.....	124
Configuring a Client/Server Architecture.....	125
Configuring Continuous Querying.....	126
Configuring Persistence.....	127
Configuring Transactions.....	128
Configuring Functions.....	128
Additional Spring Data GemFire Examples.....	130

## **Part IV: Basic Configuration and Programming.....131**

### **Chapter 19: Distributed System and Cache Configuration.....133**

Distributed System Members.....	133
Options for Configuring the Distributed System.....	134
Options for Configuring the Cache and Data Regions .....	135
Local and Remote Membership and Caching.....	135

### **Chapter 20: Cache Management.....137**

Introduction to Cache Management.....	137
Managing a Peer or Server Cache.....	139
Managing a Client Cache.....	139
Managing a Cache in a Secure System.....	140
Managing RegionServices for Multiple Secure Users.....	141
Launching an Application after Initializing the Cache.....	142

---

<b>Chapter 21: Data Regions.....</b>	<b>145</b>
Data Region Management.....	145
Creating a Region with gfsh.....	148
Creating a Region Through the cache.xml File.....	148
Creating a Region Through the API.....	149
Region Naming.....	150
Region Shortcuts and Custom Named Region Attributes.....	150
Storing and Retrieving Region Shortcuts and Custom Named Region Attributes.....	152
Managing Region Attributes.....	153
Creating Custom Attributes for Regions and Entries.....	155
<b>Chapter 22: Data Entries .....</b>	<b>157</b>
Managing Data Entries.....	157
Requirements for Using Custom Classes in Data Caching.....	159
<b>Part V: Topologies and Communication.....</b>	<b>161</b>
<b>Chapter 23: Topology and Communication General Concepts.....</b>	<b>163</b>
Topology Types.....	163
Planning Topology and Communication .....	164
How Member Discovery Works.....	165
How Communication Works.....	167
Using Bind Addresses.....	169
Choosing Between IPv4 and IPv6.....	171
<b>Chapter 24: Peer-to-Peer Configuration.....</b>	<b>173</b>
Configuring Peer-to-Peer Discovery.....	173
Configuring Peer Communication.....	174
Organizing Peers into Logical Member Groups.....	174
<b>Chapter 25: Client/Server Configuration.....</b>	<b>177</b>
Standard Client/Server Deployment.....	177
How Server Discovery Works.....	178
How Client/Server Connections Work.....	180
Configuring a Client/Server System.....	183
Organizing Servers Into Logical Member Groups.....	184
Client/Server Example Configurations.....	184
Fine-Tuning Your Client/Server Configuration.....	187
<b>Chapter 26: Multi-site (WAN) Configuration.....</b>	<b>189</b>
How Multi-site (WAN) Systems Work.....	189
Multi-site (WAN) Topologies.....	193
Configuring a Multi-site (WAN) System.....	195
Filtering Events for Multi-Site (WAN) Distribution.....	199

Resolving Conflicting Events.....	200
<b>Part VI: Developing with vFabric GemFire.....</b>	<b>203</b>
<b>Chapter 27: Region Data Storage and Distribution.....</b>	<b>205</b>
Storage and Distribution Options.....	205
Region Types.....	206
Region Data Stores and Data Accessors.....	208
Creating Regions Dynamically.....	208
<b>Chapter 28: Partitioned Regions.....</b>	<b>213</b>
Understanding Partitioning.....	213
Configuring Partitioned Regions.....	215
Configuring the Number of Buckets for a Partitioned Region.....	216
Custom-Partitioning and Co-locating Data.....	217
Configuring High Availability for Partitioned Regions.....	225
Configuring Single-Hop Client Access to Server-Partitioned Regions.....	229
Rebalancing Partitioned Region Data.....	230
<b>Chapter 29: Distributed and Replicated Regions.....</b>	<b>233</b>
How Distribution Works.....	233
Options for Region Distribution.....	234
How Replication and Preloading Work.....	235
Configure Distributed, Replicated, and Preloaded Regions.....	236
Locking in Global Regions.....	237
<b>Chapter 30: Consistency for Region Updates.....</b>	<b>239</b>
Consistency Checking by Region Type.....	239
Configuring Consistency Checking.....	240
Overhead for Consistency Checks.....	241
How Consistency Checking Works for Replicated Regions.....	241
How Destroy and Clear Operations Are Resolved.....	243
Transactions with Consistent Regions.....	243
How Consistency Is Achieved in WAN Deployments.....	244
<b>Chapter 31: General Region Data Management.....</b>	<b>245</b>
Persistence and Overflow.....	245
Eviction.....	248
Expiration.....	250
Keeping the Cache in Sync with Outside Data Sources.....	253
<b>Chapter 32: Data Serialization.....</b>	<b>257</b>
Overview of Data Serialization.....	257
vFabric GemFire PDX Serialization.....	258
vFabric GemFire Data Serialization (DataSerializable and DataSerializer).....	272

---

Standard Java Serialization.....	273
<b>Chapter 33: Events and Event Handling.....</b>	<b>275</b>
How Events Work.....	275
Implementing GemFire Event Handlers.....	288
Configuring Peer-to-Peer Event Messaging.....	297
Configuring Client/Server Event Messaging .....	297
Configuring Event Queues.....	309
<b>Chapter 34: Delta Propagation.....</b>	<b>317</b>
How Delta Propagation Works.....	317
When to Avoid Delta Propagation.....	319
Delta Propagation Properties.....	319
Implementing Delta Propagation.....	321
Errors In Delta Propagation.....	322
Delta Propagation Example.....	322
<b>Chapter 35: Querying.....</b>	<b>325</b>
GemFire Querying FAQ and Examples.....	325
Basic Querying.....	336
Advanced Querying.....	356
<b>Chapter 36: Continuous Querying.....</b>	<b>375</b>
How Continuous Querying Works.....	375
Implementing Continuous Querying.....	377
Managing Continuous Querying.....	381
<b>Chapter 37: Transactions.....</b>	<b>383</b>
About Transactions.....	383
Types of Transactions.....	383
GemFire Cache Transactions.....	384
JTA Global Transactions with GemFire.....	401
Monitoring and Troubleshooting Transactions.....	409
Transaction Coding Examples.....	410
<b>Chapter 38: Function Execution.....</b>	<b>417</b>
Use Cases for Function Execution.....	417
How Function Execution Works.....	417
Executing a Function in vFabric GemFire.....	421
<b>Part VII: Managing vFabric GemFire.....</b>	<b>427</b>
<b>Chapter 39: GemFire Management and Monitoring.....</b>	<b>429</b>
Management and Monitoring Features.....	429
Architecture and Components.....	430

JMX Manager Operations.....	.433
Federated MBean Architecture.....	.437
Configuring RMI Registry Ports and RMI Connectors.....	.446
Executing gfsh Commands through the Management API.....	.446
Management and Monitoring Programming Examples.....	.447
<b>Chapter 40: Heap Use and Management.....</b>	<b>453</b>
How the Resource Manager Works.....	.453
Control Heap Use with the Resource Manager.....	.454
Resource Manager Example Configurations.....	.457
<b>Chapter 41: Disk Storage.....</b>	<b>459</b>
How Disk Stores Work.....	.459
Disk Store File Names and Extensions.....	.460
Disk Store Operation Logs.....	.462
Configuring Disk Stores.....	.463
Optimizing a System with Disk Stores.....	.468
Starting and Shutting Down a System with Disk Stores.....	.468
Disk Store Management.....	.471
Back Up and Restore a Disk Store.....	.478
<b>Chapter 42: Cache and Region Snapshots.....</b>	<b>483</b>
Usage and Performance Notes.....	.483
Exporting Cache and Region Snapshots.....	.484
Importing Cache and Region Snapshots.....	.485
Filtering Entries During Import or Export.....	.486
Reading Snapshots Programmatically.....	.486
<b>Chapter 43: Network Partitioning.....</b>	<b>489</b>
How Network Partitioning Management Works.....	.489
Failure Detection and Membership Views.....	.490
Membership Coordinators, Lead Members and Member Weighting.....	.491
Network Partitioning Scenarios.....	.493
Configure vFabric GemFire to Handle Network Partitioning.....	.494
Preventing Network Partitions.....	.495
<b>Chapter 44: Security.....</b>	<b>497</b>
vFabric GemFire Security Features.....	.497
Security Information Specific to vFabric GemFire.....	.497
Security Implementation Overview.....	.499
Where to Place Security Settings.....	.500
Authentication.....	.500
Authorization.....	.509
SSL.....	.513

---

<b>Chapter 45: Performance Tuning and Configuration.....</b>	<b>517</b>
Improving GemFire Performance on vSphere.....	517
Performance Controls.....	520
System Member Performance.....	524
Slow Receivers with TCP/IP.....	525
Slow distributed-ack Messages.....	530
Socket Communication.....	530
UDP Communication.....	536
Multicast Communication.....	537
Maintaining Cache Consistency.....	542
Logging.....	543
<b>Chapter 46: Statistics.....</b>	<b>551</b>
How Statistics Work.....	551
Transient Region and Entry Statistics.....	551
Application-Defined and Custom Statistics.....	552
Configuring and Using Statistics.....	553
GemFire Statistics List.....	555
<b>Chapter 47: Troubleshooting and System Recovery.....</b>	<b>605</b>
Producing Artifacts for Troubleshooting.....	605
Diagnosing System Problems.....	606
System Failure and Recovery.....	616
Recovering from Application and Cache Server Crashes.....	621
Recovering from Machine Crashes.....	626
Understanding and Recovering from Network Outages.....	627
<b>Part VIII: Deploying and Running vFabric GemFire.....</b>	<b>629</b>
<b>Chapter 48: Deploying vFabric GemFire Configuration Files.....</b>	<b>631</b>
Main Steps to Deploying Configuration Files.....	631
Default File Specifications and Search Locations.....	631
Changing the File Specifications.....	632
Deploying Configuration Files in JAR Files.....	633
<b>Chapter 49: Managing vFabric GemFire System Output Files.....</b>	<b>635</b>
<b>Chapter 50: Starting Up and Shutting Down Your System.....</b>	<b>637</b>
Introduction to Startup and Shutdown.....	637
Starting Up Your System.....	637
Shutting Down the System.....	638
<b>Chapter 51: Using the GemFire SHell (gfsh).....</b>	<b>641</b>
What You Can Do with gfsh.....	641

Starting gfsh.....	641
Basic Shell Features and Command-Line Usage.....	642
Tutorial: Performing Common Tasks with gfsh.....	646
Configuring the gfsh Environment.....	652
Useful gfsh Shell Variables.....	654
Quick Reference of gfsh Commands by Functional Area.....	655
gfsh Command Help.....	673
Creating and Running gfsh Command Scripts.....	733
<b>Chapter 52: vFabric GemFire Locator Process.....</b>	<b>737</b>
<b>Chapter 53: vFabric GemFire Servers.....</b>	<b>741</b>
<b>Chapter 54: Using Member Groups.....</b>	<b>745</b>
<b>Chapter 55: Deploying Application JARs to vFabric GemFire Members.</b>	<b>747</b>
<b>Chapter 56: Firewall Considerations.....</b>	<b>751</b>
Firewalls and Connections.....	751
Firewalls and Ports.....	751
Locators, Gateway Receivers, and Ports.....	753
<b>Part IX: Tools and Modules.....</b>	<b>755</b>
<b>Chapter 57: HTTP Session Management Modules.....</b>	<b>757</b>
HTTP Session Management Quick Start.....	757
Advantages of Using GemFire for Session Management.....	759
Common Topologies for HTTP Session Management.....	761
General Information on HTTP Session Management.....	762
Session State Log Files.....	763
HTTP Session Management Module for vFabric tc Server.....	765
HTTP Session Management Module for Tomcat.....	776
HTTP Session Management Module for AppServers.....	786
<b>Chapter 58: Hibernate Cache Module.....</b>	<b>801</b>
Why Use Gemfire with Hibernate?.....	801
How GemFire Works with Hibernate.....	801
Installing the Hibernate Cache Module.....	802
Setting Up the GemFire Hibernate Cache Module.....	802
Advanced Configuration (Hibernate Cache Module).....	804
Changing vFabric GemFire Default Configuration (Hibernate Cache Module).....	807
Using Hibernate Cache Module with HTTP Session Management Module.....	811
<b>Chapter 59: Gemcached.....</b>	<b>813</b>
How Gemcached Works.....	813

---

Deploying and Configuring a Gemcached Server.....	814
Advantages of Gemcached over Memcached.....	816
<b>Chapter 60: DiskConverterPre65to65.....</b>	<b>817</b>
Running DiskConverterPre65to65.....	817
<b>Chapter 61: GemFire Pulse.....</b>	<b>819</b>
Pulse Quick Start (Embedded Mode).....	819
Hosting Pulse on a Web Application Server.....	820
Configuring Pulse Authentication.....	822
Using Pulse Views.....	822
<b>Chapter 62: DataBrowser.....</b>	<b>827</b>
DataBrowser System Requirements.....	827
Installation and Licensing.....	827
Running DataBrowser.....	828
Connecting DataBrowser to a vFabric GemFire System.....	829
Tour of the DataBrowser Interface.....	830
DataBrowser Preferences.....	833
How DataBrowser Works.....	835
<b>Chapter 63: Visual Statistics Display.....</b>	<b>839</b>
VSD Overview.....	839
VSD System Requirements.....	841
Installing and Running VSD.....	842
Viewing Statistics in VSD.....	843
<b>Part X: Reference.....</b>	<b>849</b>
<b>Chapter 64: gemfire.properties and gfsecurity.properties (GemFire Properties).851</b>	
Using Non-ASCII Strings in vFabric GemFire Property Files.....	860
<b>Chapter 65: cache.xml .....</b>	<b>861</b>
cache.xml Quick Reference.....	861
<cache> Element Hierarchy.....	862
<cache> Element Reference.....	865
<client-cache> Element Hierarchy .....	912
<client-cache> Element Reference.....	915
<b>Chapter 66: Region Shortcuts.....</b>	<b>959</b>
Region Shortcuts Quick Reference.....	959
Region Shortcuts Reference.....	963
<b>Chapter 67: Exceptions and System Failures.....</b>	<b>987</b>

<b>Chapter 68: Memory Requirements for Cached Data.....</b>	<b>989</b>
<b>Glossary.....</b>	<b>993</b>



# Technical Support

VMware provides several sources for product information and support. The *vFabric GemFire User's Guide*, the *vFabric GemFire Native Client User's Guide*, the *vFabric GemFire Tools Guide* and the online API documentation sets should always be your first source of information. VMware 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 technical support.
- You want to report a bug.
- You want to submit a feature request.

For questions concerning product availability, pricing, license serial numbers, or future features, contact your account manager.

## Preserving Artifacts for Technical Support

If you have a hung application and you do not have to kill it, leave it while you contact technical support. If you cannot leave the application 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 application'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 information.
- 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 VMware support sites at <https://www.vmware.com/support/contacts/> and <https://www.vmware.com/support/policies/howto.html> provide all the information you need to contact our technical support team. To file a support request, use the VMware technical support request site at <http://www.vmware.com/srfile/>. Note that you will need a valid VMware support profile account to be able to login and file the support request. You can also use the system to view your active support contract levels. If you are unable to access the website for any reason, you can call technical support (U.S. English/Toll Free) at 1-877-486-9273.

When contacting VMware technical support, please be prepared to provide this information:

- Your name, company name, and GemFire license numbers
- 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 you are able to gather

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

### **24x7 Emergency Technical Support**

VMware 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 account manager for more details.

### **Training and Consulting**

Consulting and training for all GemFire products are available through VMware's professional services organization.

Training courses are offered periodically at VMware'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 our products in your business environment.

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

# About vFabric GemFire User's Guide

Revised February 18, 2014.

*vFabric GemFire User's Guide* provides step-by-step procedures for installation, configuration, deployment, management and troubleshooting of VMware® vFabric™ GemFire® data management system. vFabric GemFire is also part of the VMware® vFabric Suite™.

## Intended Audience

*vFabric GemFire User's Guide* is for anyone who wants to install or deploy a vFabric GemFire system, including architects planning to use GemFire, developers programming their applications to use GemFire, and administrators who need to deploy and manage a GemFire distributed caching system. It is assumed that readers have experience developing with Java.



**Note:** All current documentation for Pivotal GemFire 7.0.2 and later is now available at <http://docs.gopivotal.com/gemfire/index.html>.



# Documentation Roadmap

The vFabric GemFire main product document consists of one document, *vFabric GemFire User's Guide*, which is organized by functional area.

- **Supported Configurations and System Requirements** Verify that your system configuration is supported and that it meets the requirements to run vFabric GemFire.
- **Getting Started with vFabric GemFire** Perform initial product and licensing setup; complete the tutorial and example walk-throughs; understand vFabric GemFire features and functionality.
- **Getting Started with Spring Data GemFire** Learn how experienced Spring developers can use Spring Data GemFire to integrate vFabric GemFire components into their applications.
- **Basic Configuration and Programming** Configure vFabric GemFire distributed system members, member caches, regions, and data entries.
- **Topologies and Communication** Learn about vFabric GemFire system topologies and implement the topology you need. Perform related tasks for using bind addresses, grouping your servers into logical groups, and balancing communication load in client/server and WAN installations.
- **Developing with vFabric GemFire** Configure and manage your data and data events. Choose a data storage model such as partitioning or replication. Manage data storage and access, data loading, data eviction and expiration, data serialization, event handling, querying, transactions, and remote function execution.
- **Managing vFabric GemFire** Perform general administration of vFabric GemFire. Set up and manage memory use, disk storage for your data and event queues, security, network partitioning, and JMX MBeans with GemFire. Monitor, tune, and troubleshoot the system.
- **Deploying and Running vFabric GemFire** Deploy, start, and stop your vFabric GemFire system. Manage your configuration and output files. Learn about using the `gfs` command line interface and the administration of cache servers and locators.
- **Tools and Modules** Learn about tools included with vFabric GemFire, such as the HTTP Session Management modules, the Hibernate Cache module, Gemcached, GemFire Pulse, DataBrowser, and VSD.
- **Reference** Review general reference information that relates to all phases of vFabric GemFire programming and deployment: properties listings, `cache.xml` elements, region shortcut descriptions and defaults, system exceptions and failures, and memory requirements for cached data.
- **Glossary** Look up terms used in this documentation.

In addition, vFabric GemFire native client users can refer to the *vFabric GemFire Native Client User's Guide*. See [vFabric GemFire Native Client User's Guide](#) for detailed documentation on how to program and configure C++ and .NET client applications that connect to GemFire servers.

For the most recent product release notes and product downloads, visit the following website:

<https://www.vmware.com/support/pubs/vfabric-gemfire.html>



# Part 1

---

## Supported Configurations and System Requirements

Revised April 22, 2014.

The sections that follow document supported operating system platforms and describe additional system requirements for vFabric GemFire.



**Note:** All current documentation for Pivotal GemFire 7.0.2 and later is available at  
<http://docs.gopivotal.com/gemfire/index.html>.

### Topics:

- [vFabric GemFire Supported Configurations](#)
- [vFabric GemFire Tools Supported Configurations](#)
- [vFabric GemFire Native Client Supported Configurations](#)
- [vFabric GemFire Modules Supported Configurations](#)



# Chapter 1

---

## vFabric GemFire Supported Configurations

The supported configurations tables are organized by the 32-bit and 64-bit platforms on which vFabric GemFire standalone runs with Level A support. For a list of platforms that are supported by vFabric GemFire as part of vFabric Suite, see [vFabric Suite supported configurations](#).

- [32-Bit Platforms](#) on page 3
- [64-Bit Platforms](#) on page 4
- [Production Support and Developer Support](#) on page 4
- [Pure Java Mode and Other Configurations](#) on page 4

Running vFabric GemFire clusters with a mix of different platforms has not been tested. We recommend that you use a consistent platform on all machines in your cluster.



**Note:** The vFabric GemFire product download does not include Java. You must download and install a supported JDK on each system running GemFire. GemFire requires the JDK (and not just a JRE) to run gfsh commands and to launch servers and locators using the ServerLauncher or LocatorLauncher APIs.

### 32-Bit Platforms

Operating System	Processor	Oracle Java SE 6, Update 26 and Later	Oracle Java SE 7, Update 5 and Later	Production or Development
RHEL 5	x86	Yes	Yes	Production
RHEL 6	x86	Yes	Yes	Production
Solaris 9	SPARC	Yes	Yes	Production
Solaris 10	SPARC	Yes	Yes	Production
SLES 10**	x86	Yes	Yes	Production
Windows 2008 Server R1**	x86	Yes	Yes	Production
Windows 2008 Server R2	x86	Yes	Yes	Production
Windows 7 Enterprise	x86	Yes	Yes	Production
Windows 7 (versions other than Enterprise)**	x86	Yes	Yes	Development

\*The Microsoft Loopback Adapter is not supported.

\*\*Indicates operating systems that were not tested during standalone vFabric GemFire product testing.

## 64-Bit Platforms

Operating System	Processor	Oracle Java SE 6, Update 26 and Later	Oracle Java SE 7, Update 5 and Later	Production or Development
RHEL 5	x64	Yes	Yes	Production
RHEL 6	x64	Yes	Yes	Production
Solaris 10	SPARC	Yes	Yes	Production
SLES 10**	x64	Yes	Yes	Production
Windows 2008 Server R1**	x64	Yes	Yes	Production
Windows 2008 Server R2	x64	Yes	Yes	Production
Windows 7 Enterprise	x64	Yes	Yes	Production
Windows 7 (versions other than Enterprise)**	x64	Yes	Yes	Development
Ubuntu 10.x**	x64	Yes	Yes	Development

\*The Microsoft Loopback Adapter is not supported.

\*\*Indicates operating systems that were not tested during vFabric GemFire standalone product testing.

## Production Support and Developer Support

The tables indicate whether the supported configuration is for production or development. Generally, production support means you can run your production application on the platform; developer support means you can develop on the platform but you should not run your production application on it.

Many developers use consumer-focused operating systems that are not certified for production use, and most vFabric products function well on popular consumer operating systems. The developer support designation is intended to convey which products are "known to work" and that VMware will provide best-effort support for resolving reported issues that are discovered. Developer Support certifications are not supported for use in production.

## Pure Java Mode and Other Configurations

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

For example, GemFire can also be installed to run in pure Java mode on any standard Java platform with some functional differences. However, this type of configuration is not tested and should not be used for production purposes without consulting with VMware support first. See [Running vFabric GemFire in Pure Java Mode](#) on page 5 for more information.

## Host Machine Requirements

This topic describes the host machine requirements for vFabric GemFire.

Each machine that will run vFabric GemFire must meet the following requirements:

- A supported Java SE installation. See [vFabric GemFire Supported Configurations](#) on page 3.

- Adequate per-user quota of file handles (ulimit for Solaris and Linux).
- TCP/IP.
- System clock set to the correct time and a time synchronization service such as Network Time Protocol (NTP). Correct time stamps allow you to do the following activities:
  - Produce logs that are useful for troubleshooting. Synchronized time stamps ensure that log messages from different hosts can be merged to reproduce a chronological history of a distributed run.
  - Aggregate product-level and application-level time statistics.
  - Accurately monitor your GemFire system with scripts and other tools that read the system statistics and log files.
- Hostname and host files that are properly configured for the machine. Hostname and host file configuration can affect gfsh and Pulse functionality. On Linux and Unix hosts, see the system manpages for more information on setting hostname and hosts.



**Note:** 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.

## Dependencies on Linux RPM Packages

vFabric GemFire has RPM package dependencies for the Red Hat Enterprise Linux (RHEL) 5 and 6 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
RHEL Server release 5 (i686)	glibc	libgcc
RHEL Server release 5 (x86_64)	glibc (i686)	libgcc (i386)
RHEL Server release 6 (i686)	glibc	libgcc
RHEL Server release 6 (x86_64)	glibc (i686)	libgcc (i386)

For versions of Linux not listed in the table, you can verify that you meet the vFabric GemFire dependencies at the library level with this 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 vFabric GemFire in Pure Java Mode

vFabric GemFire can run in pure Java mode, meaning GemFire runs without the GemFire native code.

Running in pure Java mode enables vFabric GemFire to run with some functional differences on unsupported platforms. Unsupported platforms include any platforms that are not listed in [vFabric GemFire Supported Configurations](#) on page 3. Running in pure Java mode should not be used for production environments without

consulting with VMware support first. In pure Java 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.

# Chapter 2

---

## vFabric GemFire Tools Supported Configurations

vFabric GemFire Tools, including GFMon, DataBrowser and VSD, are supported on a variety of platforms.

### GemFire Pulse System Requirements

Verify that your system meets the installation and runtime requirements for GemFire Pulse.

The GemFire Pulse Web application has been tested for compatibility with the following Web browsers. GemFire Pulse may work with other operating systems and browsers upon which it has not been tested:

- Internet Explorer 9.0.8112.16421
- Safari 5.1.7 for Windows
- Google Chrome 22.0.1229.79 m
- Mozilla Firefox 16.0.1

GemFire Pulse has been tested for standalone deployment on Tomcat and Jetty.

If you are interested in using a platform that is not listed, please contact your VMware sales representative. VMware will evaluate whether it can support the platform, either as-is or under a special agreement, or whether there are reasons that prevent deploying on a particular platform.

### DataBrowser System Requirements

Verify that your system meets the installation and runtime requirements for DataBrowser.

DataBrowser is designed to run on Microsoft Windows and Red Hat Linux. DataBrowser requires the Java Runtime Environment (JRE) 1.6.0 or higher to run.

You must use the vFabric GemFire DataBrowser executable included with GemFire 7.0 to connect to vFabric GemFire 7.0 distributed systems. You cannot use past versions of vFabric GemFire DataBrowser to connect to a GemFire 7.0 system.

The minimum recommended hardware is a machine with Intel® 2 GHz Pentium® 4 (or equivalent) and 2 GB RAM.

In the DataBrowser startup scripts, you may want to set the JVM heap size explicitly to a value suitable for the typical result sizes returned by the queries that you run in your distributed system.

### VSD System Requirements

View a list of platforms that are known to work with VSD.

The following platforms are known to work with the current VSD release:

- RedHat 3
- RedHat 4
- RedHat 5
- RedHat 6
- SLES 8
- SLES 9
- SLES 10
- Solaris 2.8
- Mac OSX
- Windows NT 4.0
- Windows 98/ME/2K
- Windows XP SP3
- Windows Server 2003
- Windows Server 2008
- Windows 7 (Note: Requires slight modification; see below.)

### **Windows 7 Support**

To use VSD on Windows 7, perform the following steps:

1. Start Windows Explorer and navigate to the `GemFireProductDir\tools\vsd\bin\` directory (where `GemFireProductDir` corresponds to the location where you installed GemFire.)
2. Right click and select properties for `vsd.bat`.
3. Select the Compatibility tab.
4. Click "Run this program in compatibility mode for" and then select Windows XP SP3.
5. Repeat steps 2 and 3 for all the other executables in the `bin` directory.

# Chapter 3

---

## vFabric GemFire Native Client Supported Configurations

Operating system requirements are listed in the chart below:

Operating System	RAM	Swap Space	32-bit Disk Space Required	64-Bit Disk Space Required
Solaris 9	2GB	256MB	73MB	81MB
Solaris 10	2GB	256MB	73MB	81MB
RHEL 5	2GB	256MB	100MB	114MB
RHEL 6	2GB	256MB	100MB	114MB
SLES 10**	2GB	256MB	100MB	114MB
Windows 2008 Server R1**	2GB	256MB	102MB	114MB
Windows 2008 Server R2	2GB	256MB	102MB	114MB
Windows 7 Enterprise	2GB	256MB	102MB	114MB

\*\*Indicates operating systems that were not tested during standalone vFabric GemFire product testing.

### Windows Support Details

The vFabric GemFire native client is built and tested on Windows 7 Enterprise.

The native client is not supported on Windows NT 4.0.

### Runtime Library Requirements

The GemFire native client also requires the Microsoft Visual C++ 2010 Redistributable Package. This package is installed for you when you use the native client .msi installer. If you do not use the .msi installer, you will need to install this package for your platform architecture manually ([x86](#) or [x64](#)) on all machines that will run the GemFire native client. This package contains runtime libraries needed by the native client.

### .NET Framework Version Support

The vFabric GemFire native client is supported with Microsoft .NET Framework 2.0, 3.0, and 3.5 and 4.0.

A Microsoft .NET Framework must be installed to support the C++/CLI (Common Language Infrastructure) library for the native client.



**Note:** You can download the .NET Framework Redistributable Package (x86 for 32-bit or x64 for 64-bit) for the version you wish to use from <http://www.microsoft.com/downloads>. If it isn't listed on the Download Center page, use the Search tool to search for ".NET Framework".

The vFabric GemFire native client 7.0 now supports .NET 4.0 and Visual Studio 2010 (for compiling C++ applications on Windows). For advantages and more information on the features of .NET 4.0 and Visual Studio 2010 SP1, see [http://msdn.microsoft.com/en-us/library/dd831853\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/dd831853(v=vs.100).aspx) and [http://msdn.microsoft.com/en-us/library/vstudio/w0x726c2\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/w0x726c2(v=vs.100).aspx).

## Linux

The vFabric GemFire native client is built on Red Hat Enterprise ES 3, kernel version 2.4.21-47.EL.

The native client is tested on the following Linux versions:

- Red Hat Enterprise 5 release 5 (Tikanga), kernel version 2.6.18-8.EL5
- Red Hat Enterprise 6, kernel version 2.6.32-220.el6.x86\_64

If you are not sure of the kernel version on your system, use this command to list it:

```
prompt> uname -r
```

The following table lists the RPM package dependencies for several Linux distributions. The i386 or i686 after the package name indicates that you must install the package for that particular architecture regardless of the native operating system architecture. All of the packages listed are available with the default media for each distribution.

**Table 1: GemFire Dependencies on Linux RPM Packages**

Linux Version	glibc	libgcc
Red Hat Enterprise Linux Server release 5 (i686)	glibc	libgcc
Red Hat Enterprise Linux Server release 5 (x86_64)	glibc (i686)	libgcc (i386)
Red Hat Enterprise Linux Server release 6 (i686)	glibc	glibc
Red Hat Enterprise Linux Server release 6 (x86_64)	glibc (i686)	ibgcc (i386)

For versions of Linux not listed in the table, you can verify that you meet the native client dependencies at the library level by using the ldd tool and entering this command:

```
prompt> ldd $GFCPP/lib/libgfccppcache.so
```

This step assumes you have already installed the native client and have set the GFCPP environment variable to *productDir*, where *productDir* represents the location of the NativeClient\_xxxx\_bnnnnn directory (xxxx is the four-digit product version and bnnnnn is the product build number).

The following libraries are external dependencies of the native library, libgfccppcache.so. Verify that the ldd tool output includes all of these:

- libdl.so.2
- libm.so.6
- libpthread.so.0
- libc.so.6
- libz.so.1

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

## Solaris

The vFabric GemFire native client is supported on the following Solaris versions:

- Solaris 9 kernel update 118558-38
- Solaris 10 kernel update 118833-24

## Running vFabric GemFire Native Client on vSphere

Without tuning, the vFabric GemFire native client can suffer a performance drop in virtual environments, including the VMWare vSphere virtual platform. Without correct tuning, you can expect to see significant performance degradation when running the GemFire native client on vSphere versus running GemFire on dedicated hardware.

We recommend that you tune your GemFire native client on vSphere deployments using the same guidelines published for vFabric GemFire.

See "Improving GemFire Performance on vSphere" in the *vFabric GemFire User's Guide* for a list of guidelines and recommendations.

vFabric GemFire native client has been certified and tested on vSphere 5.0.

## Software Requirements for Using SSL

If you plan on using SSL in your GemFire native client and server deployment, you will need to download and install OpenSSL.

The GemFire native client requires OpenSSL 1.0.1g or later for Unix-based platforms. For Windows platforms, you can use OpenSSL 1.0.1g or the OpenSSL 1.0.1g "Light" version.

In addition, make sure that your system environment variables have been configured to include OpenSSL.

See "SSL Client/Server Communications" in the *vFabric GemFire Native Client User's Guide* for instructions.



# Chapter 4

---

## vFabric GemFire Modules Supported Configurations

The sections that follow document supported operating system platforms and describe additional system requirements for vFabric GemFire.

### HTTP Session Management Module Supported Configurations

vFabric GemFire HTTP Session Management modules support the following application servers and their versions.

Supported Application Server	Supported Version	vFabric GemFire Module Documentation
tc Server	2.5	<a href="#">HTTP Session Management Module for vFabric tc Server on page 765</a>
tc Server	2.6	<a href="#">HTTP Session Management Module for vFabric tc Server on page 765</a>
tc Server	2.7	<a href="#">HTTP Session Management Module for vFabric tc Server on page 765</a>
tc Server	2.8	<a href="#">HTTP Session Management Module for vFabric tc Server on page 765</a>
Tomcat	6.0	<a href="#">HTTP Session Management Module for Tomcat on page 776</a>
Tomcat	7.0	<a href="#">HTTP Session Management Module for Tomcat on page 776</a>
WebLogic	11g (10.3.x)	<a href="#">HTTP Session Management Module for AppServers on page 786</a>
WebSphere	7, 8	<a href="#">HTTP Session Management Module for AppServers on page 786</a>
JBoss	5, 6, 7	<a href="#">HTTP Session Management Module for AppServers on page 786</a>



**Note:** This table lists all application server versions that have been tested with the GemFire HTTP Session Management modules. However, the generic HTTP Session Management Module for AppServers is implemented as a servlet filter and should work on any application server platform that supports the Java Servlet 2.4 specification.

For more information on how to install and get started with the HTTP Session Management Modules, see [HTTP Session Management Quick Start on page 757](#).

### Hibernate Module Supported Configurations

vFabric GemFire supports Hibernate 3.3 and later (up to version 3.6.10) for use with the Hibernate Cache module.



**Note:** GemFire does not currently support Hibernate 4.x for use with the Hibernate Cache module.

For more information on using the Hibernate Module, see "Hibernate Cache Module" in the *vFabric GemFire User's Guide*.

# Part 2

---

## Getting Started with vFabric GemFire

*Getting Started with vFabric GemFire* provides step-by-step procedures for initial installation and configuration of vFabric GemFire. Licensing concepts and license installation procedures are provided. Tutorials and examples demonstrate product features; a documentation roadmap summarizes each main part of the User's Guide; and a main features section describes key functionality.

### Topics:

- [vFabric GemFire in 15 Minutes or Less](#)
- [About vFabric GemFire](#)
- [Main Features of vFabric GemFire](#)
- [Installing vFabric GemFire](#)
- [Upgrading vFabric GemFire](#)
- [Understanding vFabric GemFire Licenses](#)
- [Installing and Configuring vFabric GemFire Licenses](#)
- [Setting Up the Product Examples](#)
- [vFabric GemFire Tutorial](#)
- [Hello World Example](#)
- [QuickStart Examples](#)



# Chapter 5

---

## vFabric GemFire in 15 Minutes or Less

Need a quick introduction to vFabric GemFire? Take this brief tour to try out basic features and functionality.

[About vFabric GemFire](#) on page 21 briefly describes some components and concepts you will explore here.

### Step 1: Download and install vFabric GemFire.

Visit the GemFire product download page, and download the latest GemFire 7.0 distribution:

[https://my.vmware.com/web/vmware/info/slug/application\\_platform/vmware\\_vfabric\\_gemfire/7\\_0](https://my.vmware.com/web/vmware/info/slug/application_platform/vmware_vfabric_gemfire/7_0)

Expand Data Management Node. Select the View Download button, and download the GemFire ZIP file. Use any graphical unzip utility or the command line to install GemFire from the ZIP file. For example:

```
unzip vFabric_GemFire_701_b40235.zip -d /home/your_userid/vfabric-gemfire
```



**Note:** If you have not done so already, download and install Java for your system. See [vFabric GemFire Supported Configurations](#) on page 3 for a list of supported Java versions. GemFire requires the JDK (and not just a JRE) to run gfsh commands and to launch servers and locators using the ServerLauncher or LocatorLauncher APIs.

### Step 2: Set GemFire, Java, and OS environment variables.

Configure GemFire and Java environment variables. For example, on Linux:

```
JAVA_HOME=/usr/java/jdk1.6.0_38
GEMFIRE=/home/your_userid/vfabric-gemfire/vFabric_GemFire_701;export GEMFIRE
GF_JAVA=$JAVA_HOME/bin/java;export GF_JAVA
PATH=$PATH:$JAVA_HOME/bin:$GEMFIRE/bin;export PATH
CLASSPATH=$CLASSPATH:
$GEMFIRE/lib/gemfire.jar:\
$GEMFIRE/lib/antlr.jar:\
$GEMFIRE/lib/gfsh-dependencies.jar:\
$GEMFIRE/lib/gfSecurityImpl.jar:\
$GEMFIRE/lib/jackson-core-asl-1.9.9.jar:\
$GEMFIRE/lib/commons-logging.jar:\
$GEMFIRE/lib/tomcat-embed-core.jar:\
$GEMFIRE/lib/tomcat-embed-logging-juli.jar:\
$GEMFIRE/lib/tomcat-embed-jasper.jar:\
$GEMFIRE/SampleCode/quickstart/classes:\
$GEMFIRE/Samplecode/tutorial/classes:\
$JAVA_HOME/lib/tools.jar;export CLASSPATH
```

See [Install vFabric GemFire from a ZIP File](#) on page 32 if you need additional guidance on setting environment variables for your platform.

### Step 3: Use gfsh to start a Locator.

Use the gfsh command line interface to start up a Locator.

1. Create a scratch working directory (for example, my\_gemfire) and change directories into it. gfsh will add Locator and server working directories and log files in this location.
2. Start gfsh by typing gfsh at the command line (or gfsh.bat if you are using Windows).

```
/ _____/ _____/ _____/ / _____/ /  
/ / ____/ / ____/ / _____/ / / _____/ /  
/ ____/ / / / _____/ / / / _____/ /  
/ _____/ / / / _____/ / / / _____/ /  
v7.0.1  
Monitor and Manage GemFire  
gfsh>
```

3. At the gfsh prompt, type:

```
gfsh>start locator --name=locator1
```

By default, gfsh starts the locator using port 10334. In addition, because no JMX Manager exists yet, gfsh starts a JMX Manager within the Locator using port 1099.

### Step 4. Start GemFire Pulse.

Start up the browser-based GemFire Pulse monitoring tool.

```
gfsh>start pulse
```

This command launches Pulse and automatically connects you to the JMX Manager running in the Locator. At the Pulse login screen, type in the default username admin and password admin.

### Step 5: Start two cache server nodes.

Start a cache server:

```
gfsh>start server --name=server1 --server-port=40405
```

This command starts a cache server named "server1" on the specified port of 40405. Next, start a second cache server. Note that this time you must specify a different --server-port argument because you are starting a cache server process on the same host machine.

```
gfsh>start server --name=server2 --server-port=40406
```

Observe the changes (new members and servers) in Pulse. Try expanding the distributed system icon to see the Locator and two cache servers graphically.

List the members of your distributed system. The locator and cache servers you just started should appear in the list.

```
gfsh>list members
```

### Step 6: Create regions.

In this step you will try out creating different types of regions with the gfsh command line utility. Regions are the core building blocks of the GemFire distributed system and provide the means for organizing your data.

1. Create a replicated region:

```
gfsh>create region --name=region1 --type=REPLICATE
```

2. Create a partitioned region:

```
gfsh>create region --name=region2 --type=PARTITION
```

3. Create a partitioned persistent region.

```
gfsh>create region --name=region3 --type=PARTITION_PERSISTENT
```

4. Use the gfsh command line to view a list of the regions that you just created.

```
gfsh>list regions
```

5. To view specifics about a region, type something like the following:

```
gfsh>describe region --name=region1
```

6. In Pulse, click the green cluster icon to see all the new members and new regions that you just added to your distributed system.



**Note:** Keep this gfsh prompt open for the next steps.

## Step 7: Begin executing the Server Managed Caching QuickStart example.



**Note:** The following steps use QuickStart examples from the *product\_install\_dir/SampleCode/quickstart* folder.

1. Keep the gfsh prompt open and start two operating system terminal sessions. Make sure each terminal session has the proper GemFire, Java and OS environment variables set (see Step 2 as a reference.)
2. In the gfsh prompt, start the QuickStart example's cache server:

```
gfsh>start server --name=server_mc  
--cache-xml-file=${SYS_GEMFIRE_DIR}/SampleCode/quickstart/xml/Server.xml
```

3. In the first terminal session, change to the *product\_install\_dir/SampleCode/quickstart* directory. Start the consumer client:

```
prompt# java quickstart.ClientConsumer all-keys
```

4. In the second terminal session, change to the *product\_install\_dir/SampleCode/quickstart* directory. Start the worker client.

```
prompt# java quickstart.ClientWorker
```

The application in this second terminal adds actual data into the region. After the client runs, do not select Enter yet!

5. Observe the changes in Pulse. For example, note that in the Subscriptions area there are now "2" subscriptions.

## Step 8: Examine the distribution of data in the region and run queries on the data.

Return to the gfsh terminal.

Use gfsh to examine the exampleRegion.

```
gfsh>describe region --name=/exampleRegion
```

Use the locate entry command to see where a particular key is located:

```
gfsh>locate entry --key=key0 --region=exampleRegion
```

Execute a query against the region:

```
gfsh>query --query="SELECT entry.value FROM /exampleRegion.entries entry WHERE entry.key='key2'"
```

### **Step 9: Stop individual members and shut down the system.**

To exit the QuickStart application and shut down your distributed system, do the following:

1. In the ClientWorker terminal session, select the Enter key to close the cache and exit the application.
2. In the second terminal, select the Enter key. The ClientConsumer application in the other terminal session will also close its cache and exit.
3. In the current gfsh session, stop the QuickStart cache server process:

```
gfsh>stop server --name=server_mc
```

You can use `stop server` and `stop locator` commands to shut down individual processes.

4. Shut down the rest of the cache servers by entering the following at the gfsh prompt:

```
gfsh>shutdown
```

When prompted, type 'Y' to confirm the shutdown of the distributed system.

5. Shut down the locator by entering the following command at the gfsh prompt:

```
gfsh>stop locator --name=locator1
```

Wait for the locator to stop.

6. To exit the gfsh prompt, type:

```
gfsh>exit
```

### **Step 10: What to do next...**

Here are some suggestions on what to explore next with vFabric GemFire:

- Try a more detailed product tutorial with a sample social networking application. See [vFabric GemFire Tutorial](#) on page 65.
- To get more practice using gfsh, see [Tutorial: Performing Common Tasks with gfsh](#) on page 646.
- If you're a Spring developer, read the section [Getting Started with Spring Data GemFire](#) on page 113.
- Go through the remaining QuickStart and code examples. See [Setting Up the Product Examples](#) on page 61 for more information.

# Chapter 6

---

## About vFabric GemFire

vFabric GemFire is a data management platform that provides real-time, consistent access to data-intensive applications throughout widely distributed cloud architectures. It is available as a standalone product and as a component of vFabric Suite.

GemFire pools memory, CPU, network resources, and optionally local disk across multiple processes to manage application objects and behavior. It uses dynamic replication and data partitioning techniques to implement high availability, improved performance, scalability, and fault tolerance. In addition to being a distributed data container, vFabric GemFire is an in-memory data management system that provides reliable asynchronous event notifications and guaranteed message delivery.

### Main Concepts and Components

*Caches* are an abstraction that describe a node in a GemFire distributed system. Application architects can arrange these nodes in several deployment architectures, including: peer to peer, client/server, and WAN gateway topologies.

Within each cache, you define data *regions*. Data regions are analogous to tables in a relational database and manage data in a distributed fashion as name/value pairs. A *replicated* region stores identical copies of the data on each cache member of a distributed system. A *partitioned* region spreads the data among cache members. After the system is configured, client applications can access the distributed data in regions without knowledge of the underlying system architecture. You can define listeners to create notifications about when data has changed, and you can define expiration criteria to delete obsolete data in a region.

GemFire uses *continuous querying* to enable event-driven architectures. GemFire ties events and data together so that when an event is processed, the data required to process the event is available without additional queries to a disk-based database. Clients can subscribe to change notifications so that they can execute tasks when a specific piece of data changes.

For large, production systems, GemFire provides *locators*. Locators provide both discovery and load balancing services. You configure clients with a list of locator services and the locators maintain a dynamic list of member servers. By default, GemFire clients and servers use port 40404 and multicast to discover each other.

For more information on product features, see [Main Features of vFabric GemFire](#) on page 23.



# Chapter 7

---

## Main Features of vFabric GemFire

This section summarizes vFabric GemFire main features and describes key functionality. For more specific information about functionality that has been added to GemFire since the 6.5 release, refer to the Release Notes.

- [Feature Summary](#) on page 23
- [High Read-and-Write Throughput](#) on page 24
- [Low and Predictable Latency](#) on page 24
- [High Scalability](#) on page 24
- [Continuous Availability](#) on page 24
- [Reliable Event Notifications](#) on page 24
- [Continuous Querying](#) on page 25
- [Parallelized Application Behavior on Data Stores](#) on page 25
- [Shared-Nothing Disk Persistence](#) on page 25
- [Multisite Data Distribution](#) on page 25
- [Reduced Cost of Ownership](#) on page 25
- [Single-Hop Capability for Client/Server](#) on page 26
- [Heterogeneous Data Sharing](#) on page 26
- [Client/Server Security](#) on page 26

### Feature Summary

GemFire includes the following features:

- Combines redundancy, WAN replication, and a "shared nothing" persistence architecture to deliver fail-safe reliability and performance.
- Continuous querying to provide active data change notifications.
- Horizontally scalable to thousands of cache members, with multiple cache topologies to meet different enterprise needs. The cache can be distributed across multiple computers.
- Asynchronous and synchronous cache update propagation.
- Delta propagation distributes only the difference between old and new versions of an object (delta) instead of the entire object, resulting in significant distribution cost savings.
- Reliable asynchronous event notifications and guaranteed message delivery through optimized, low latency distribution layer.
- Applications run 4 to 40 times faster with no additional hardware.
- Data awareness and real-time business intelligence. If data changes as you retrieve it, you see the changes immediately.
- Integration with Spring Framework to speed and simplify the development of scalable, transactional enterprise applications.
- JTA compliant transaction support.
- Native support for Java, C++, and C# applications.

## High Read-and-Write Throughput

GemFire uses concurrent main-memory data structures and a highly optimized distribution infrastructure to provide more than 10 times the read-and-write throughput of traditional disk-based databases. Applications can make copies of data dynamically in memory through synchronous or asynchronous replication for high read throughput, or partition the data across many GemFire system members to achieve high read-and-write throughput. Data partitioning doubles the aggregate throughput if the data access is fairly balanced across the entire data set. Linear increase in throughput is limited only by the backbone network capacity.

## Low and Predictable Latency

GemFire's optimized caching layer minimizes context switches between threads and processes. It manages data in highly concurrent structures to minimize contention points. Communication to peer members is synchronous if the receivers can keep up, which keeps the latency for data distribution to a minimum. Servers manage object graphs in serialized form to reduce the strain on the garbage collector.

GemFire partitions subscription management (interest registration and continuous queries) across server data stores, ensuring that a subscription is processed only once for all interested clients. The resulting improvements in CPU use and bandwidth utilization improve throughput and reduce latency for client subscriptions.

## High Scalability

GemFire achieves scalability through dynamic partitioning of data across many members and spreading the data load uniformly across the servers. For "hot" data, you can configure the system to expand dynamically to create more copies of the data. You can also provision application behavior to run in a distributed manner in close proximity to the data it needs.

If you need to support high and unpredictable bursts of concurrent client load, you can increase the number of servers managing the data and distribute the data and behavior across them to provide uniform and predictable response times. Clients are continuously load balanced to the server farm based on continuous feedback from the servers on their load conditions. With data partitioned and replicated across servers, clients can dynamically move to different servers to uniformly load the servers and deliver the best response times.

You can also improve scalability by implementing asynchronous "write behind" of data changes to external data stores like a database. GemFire avoids a bottleneck by queuing all updates in order and redundantly. You can also conflate updates and propagate them in batch to the database.

## Continuous Availability

In addition to guaranteed consistent copies of data in memory, applications can persist data to disk on one or more GemFire members synchronously or asynchronously, by using GemFire's "shared nothing disk architecture". All asynchronous events (store-forward events) are redundantly managed in at least two members such that if one server fails, the redundant one takes over. All clients connect to logical servers, and the client fails over automatically to alternate servers in a group during failures or when servers become unresponsive.

## Reliable Event Notifications

Publish/subscribe systems offer a data distribution service where new events are published into the system and routed to all interested subscribers in a reliable manner. Traditional messaging platforms focus on message delivery, but often the receiving applications need access to related data before they can process the event. This often requires them to access a standard database when the event is delivered, making the subscriber limited by the speed of the database.

GemFire offers data and events through a single system. Data is managed as objects in one or more distributed data regions, similar to tables in a database. Applications simply insert, update, or delete objects in data regions, and the platform delivers the object changes to the subscribers. The subscriber receiving the event has direct access to the related data in local memory or can fetch the data from one of the other members through a single hop.

## Continuous Querying

In messaging systems like JMS, clients subscribe to topics and queues. Any message delivered to a topic is sent to the subscriber. GemFire allows applications to express complex interest using the OQL query language and referred to as continuous querying.

## Parallelized Application Behavior on Data Stores

You can execute application business logic in parallel on the GemFire members. GemFire's data-aware function execution service permits execution of arbitrary data-dependent application functions on the members where the data is partitioned for locality of reference and scale.

By co-locating the relevant data and parallelizing the calculation, you dramatically increase overall throughput. More importantly, the calculation latency is inversely proportional to the number of members on which it can be parallelized.

The fundamental premise is to route the function transparently to the application that carries the data subset required by the application function and avoid moving data around on the network. Application function can be executed on only one member, executed in parallel on a subset of members or in parallel across all members. This programming model is similar to the popular Map-Reduce model from Google. Data-aware function routing is most appropriate for applications that require iteration over multiple data items (such as a query or custom aggregation function).

## Shared-Nothing Disk Persistence

Each GemFire system member manages data on disk files independent of other members. Failures in disks or cache failures in one member do not affect the ability of another cache instance to operate safely on its disk files. This "shared nothing" persistence architecture allows applications to be configured such that different classes of data are persisted on different members across the system, dramatically increasing the overall throughput of the application even when disk persistence is configured for application objects.

Unlike a traditional database system, GemFire does not manage data and transaction logs in separate files. All data updates are appended to files that are similar to transactional logs of traditional databases. You can avoid disk seek times if the disk is not concurrently used by other processes, and the only cost incurred is the rotational latency. Even some of the best disk technology today takes about 2ms to seek to a track.

## Multisite Data Distribution

Scalability problems can result from data sites being spread out geographically across a WAN. GemFire offers a novel model to address these topologies, ranging from a single peer-to-peer cluster to reliable communications between data centers across the WAN. This model allows distributed systems to scale out in an unbounded and loosely-coupled fashion without loss of performance, reliability or data consistency.

At the core of this architecture is the gateway sender configuration used for distributing region events to a remote site. You can deploy gateway sender instances in parallel, which enables GemFire to increase the throughput for distributing region events across the WAN. You can also configure gateway sender queues for persistence and high availability to avoid data loss in the case of a member failure.

## Reduced Cost of Ownership

You can configure caching in tiers. The client application process can host a cache locally (in memory and overflow to disk) and delegate to a cache server farm on misses. Even a 30 percent hit ratio on the local cache translates to significant savings in costs. The total cost associated with every single transaction comes from the CPU cycles spent, the network cost, the access to the database, and intangible costs associated with database maintenance. By managing the data as application objects, you avoid the additional cost (CPU cycles) associated with mapping SQL rows to objects.

## Single-Hop Capability for Client/Server

Clients can send individual data requests, such as put and delete, in a single hop, directly to the server holding the data key. The clients store metadata about bucket locations for partitioned region data. This feature improves performance and client access to partitioned regions in the server tier.

## Heterogeneous Data Sharing

C#, C++ and Java applications can share application business objects without going through a transformation layer such as SOAP or XML. The server side behavior, though implemented in Java, provides a unique native cache for C++ and .NET applications. Application objects can be managed in the C++ process heap and distributed to other processes using a common "on-the-wire" representation for objects. A C++ serialized object can be directly deserialized as an equivalent Java or C# object. A change to a business object in one language can trigger reliable notifications in applications written in the other supported languages.

## Client/Server Security

GemFire supports running multiple, distinct users in client applications. This feature accommodates installations where GemFire clients are embedded in application servers, and each application server supports data requests from many users. Each user may be authorized to access a small subset of data on the servers, as in a customer application where each customer can access only their own orders and shipments. Each user in the client connects to the server with its own set of credentials, and is given its own access authorization to the server cache.

Client/server communication has increased security against replay attacks. The server now sends the client a unique, random identifier with each response, to be used in the next client request. Because of the identifier, even a repeated client operation call is sent as a unique request to the server.

# Chapter 8

---

## Installing vFabric GemFire

How you install vFabric GemFire depends on your operating system and on whether you buy GemFire as part of vFabric Suite or as a standalone product.

### Pre-Installation Note for vFabric Suite Customers

If you obtain vFabric GemFire as part of a vFabric Suite package, you install it exclusively on VMware virtual machines that run on vSphere, and you activate your vFabric Suite license through the vFabric License Server and vCenter server.

If you obtain GemFire as part of vFabric Suite, first complete the vFabric License Server installation and vFabric Suite license activation procedures in [Getting Started with vFabric Suite](#). Then follow procedures in [Installing vFabric GemFire](#) on page 27 and [vFabric Suite Only: Configure GemFire for vFabric License Server](#) on page 58.

### RHEL: Install vFabric GemFire (Standalone) from RPM

If your guest operating system is Red Hat Enterprise Linux (RHEL), VMware recommends that you use the RPM distribution to install vFabric GemFire. You complete the installation procedure on every virtual and physical machine that will run GemFire.

When you install vFabric GemFire on RHEL from RPM, the software is installed by default into `/opt/vmware/vfabric-gemfire/vFabric_GemFire_XXX` where XX corresponds to the version of GemFire (for example, vFabric\_GemFire\_701) that you have installed. The vFabric GemFire installation is owned by the `gemfire` user in the `vfabric` group.

### Prerequisites

The following tasks are prerequisites for installing vFabric GemFire from RPM:

- Confirm that your system meets the hardware and software requirements described in [Supported Configurations and System Requirements](#) on page 1.
- From the [VMware downloads page](#), select **VMware vFabric GemFire**. On the GemFire 7.0 download page, perform the following steps:
  - If you are installing GemFire for evaluation, click on the "**Download Free Trial**" link, register an account with VMware (if you have not done so already) and download GemFire.
  - If you have already purchased GemFire, locate the vFabric GemFire licensed offering (Data Management Node, Application Cache Node, Global WAN Upgrade or Unlimited Client Upgrade) that you have purchased from the Product Downloads tab. Click on the **View Download** button.



**Note:** You can also get GemFire from your salesperson.

- Download the vFabric GemFire RPM appropriate for your RHEL operating system.
- RHEL 5: `vfabric-gemfire-7.0.1-2.el5.noarch.rpm`
- RHEL 6: `vfabric-gemfire-7.0.1-2.el6.noarch.rpm`

## Procedure

Use the following procedure to install vFabric GemFire from RPM:

1. From the RHEL computer on which you will install vFabric GemFire, log in as the root user or as an unprivileged user.
2. Execute the following `rpm` command. If necessary, use `sudo` to run the command either as root or as the `gemfire` user. For example:

### RHEL 5:

```
prompt# sudo -u gemfire -E rpm -ivh vfabric-gemfire-7.0.1-2.el5.noarch.rpm
```

### RHEL 6:

```
prompt# sudo -u gemfire -E rpm -ivh vfabric-gemfire-7.0.1-2.el6.noarch.rpm
```

The `rpm` command begins the install process, resolves dependencies, and displays the packages it plans to install.

3. If the installation is successful, you see a [100%] installation status message. For example:

Preparing... [100%] 1:vfabric-gemfire [100%]	#####
---	-------

The `rpm` command:

- Installs vFabric GemFire into the `/opt/vmware/vfabric-gemfire/` directory.
- If the user does not already exist, adds a `gemfire` non-interactive user (in a group called `vfabric`) and sets the owner of all directories and files under `/opt/vmware/vfabric-gemfire/` to `gemfire`.

Note that you cannot login directly as the `gemfire` user because interactive login has been disabled. Rather, you must login as the root user or as a privileged user using `sudo`, and then `su - gemfire`.

- Sets up the commands listed in [Commands and Services Installed by the RPM](#) on page 29.

  4. Configure your environment for GemFire.

- Set the `GEMFIRE` environment variable to point to your GemFire installation top-level directory. (You should see `bin`, `lib`, `dtd`, and other directories under `GEMFIRE`.) For example:

```
GEMFIRE=/opt/vmware/vfabric-gemfire/vFabric_GemFire_701
export GEMFIRE
```

- Configure `GF_JAVA` and your `PATH` and `CLASSPATH` as shown in these examples. `GF_JAVA` must point to the `java` executable file under your `JAVA_HOME`.

- Set the `GEMFIRE` environment variable to point to your GemFire installation top-level directory. The RPM will install GemFire in the following directory:

```
GEMFIRE=/opt/vmware/vfabric-gemfire/vFabric_GemFire_701
export GEMFIRE
```

- Configure your `PATH`, `CLASSPATH` and `GF_JAVA` environment variables as shown in these examples. `GF_JAVA` must point to the `java` executable file under your `JAVA_HOME`. (If you have not done so already, you should also set your `JAVA_HOME` variable to a supported Java installation.)

If you will be using the gfsh command-line utility or managing servers and locators with the ServerLauncher and LocatorLauncher APIs, then you must set JAVA\_HOME to a JDK installation and include tools.jar in your CLASSPATH. For example:

```
JAVA_HOME=/usr/java/jdk1.6.0_38
export JAVA_HOME
GF_JAVA=$JAVA_HOME/bin/java
export GF_JAVA
PATH=$PATH:$JAVA_HOME/bin:$GEMFIRE/bin
export PATH
CLASSPATH=$CLASSPATH:$GEMFIRE/lib/gemfire.jar:\
$GEMFIRE/lib/antlr.jar:\
$GEMFIRE/lib/gfsh-dependencies.jar:\
$GEMFIRE/lib/gfSecurityImpl.jar:\
$GEMFIRE/lib/jackson-core-asl-1.9.9.jar:\
$GEMFIRE/lib/commons-logging.jar:\
$GEMFIRE/lib/tomcat-embed-core.jar:\
$GEMFIRE/lib/tomcat-embed-logging-juli.jar:\
$GEMFIRE/lib/tomcat-embed-jasper.jar:\
$JAVA_HOME/lib/tools.jar
export CLASSPATH
```



**Note:** You only need to add commons-logging.jar, tomcat-embed-core.jar, tomcat-embed-logging-juli.jar, and tomcat-embed-jasper.jar files to your CLASSPATH if you are planning on using the GemFire member as a GemFire Pulse application server.

5. If you have purchased a production license, activate the license. See [Understanding vFabric GemFire Licenses](#) on page 45 and [Installing and Configuring vFabric GemFire Licenses](#) on page 55.
6. Repeat this procedure for every virtual or physical machine on which you will run vFabric GemFire.

## Commands and Services Installed by the RPM

When you install vFabric GemFire from RPM, the RPM also installs the following:

- Installs a cacheserver service. This allows you to manage a default cache server with the following commands:

```
/sbin/service cacheserver start
/sbin/service cacheserver stop
/sbin/service cacheserver restart
```

- Creates the symlink in /usr/bin/gfsh that points to the installed location of gfsh. This way the user can execute gfsh from any directory.



**Note:** The installation of these commands and services is only available when installing from RPM. See [RHEL Only: Using the cacheserver Service](#) on page 743 for more information on using the service.

## What to Do Next

After you install vFabric GemFire from RPM, you perform the following tasks:

- If you obtained vFabric GemFire as a standalone product, install a license (serial number) on each physical and virtual machine on which you installed vFabric GemFire. See [Understanding vFabric GemFire Licenses](#) on page 45 and [Installing and Configuring vFabric GemFire Licenses](#) on page 55. (Note that you do not need to install any vFabric GemFire licenses on any machines that are only running client applications.)
- Run the product tutorial and examples. See [Setting Up the Product Examples](#) on page 61 and [vFabric GemFire Tutorial](#) on page 65.

- If you need to uninstall GemFire, see [How to Uninstall GemFire](#) on page 34.

## RHEL: Install vFabric GemFire (vFabric Suite) from the VMware YUM Repository

If your guest operating system is Red Hat Enterprise Linux (RHEL) and you are installing from the VMware RPM repository, VMware recommends that you use yum to install vFabric GemFire. You complete the installation procedure on every virtual and physical machine that will run GemFire.

When you install vFabric components on RHEL from the VMware RPM repository, the components are installed into different directories and are owned by different users in different groups. The vFabric GemFire installation is owned by the `gemfire` user in the `vfabric` group. The default installation directory is `/opt/vmware/vfabric-gemfire/vFabric_GemFire_XXX` where `XXX` corresponds to the version of GemFire (for example, `vFabric_GemFire_701`) that you have installed.

### Prerequisites

- Confirm that your system meets the hardware and software requirements described in [Supported Configurations and System Requirements](#) on page 1.
- Install the vFabric 5 RPMs (you do this procedure only once for the entire installation):
  1. Log in to the RHEL VM as the root user (or as an unprivileged user who has sudo privileges) and start a terminal.
  2. Use the `yum search vfabric` command to view the list of vFabric components that you can install from the VMware repository. The vFabric GemFire RPM is called `vfabric-gemfire`.

### Procedure

1. From the RHEL computer on which you will install vFabric GemFire, log in as the root user or as an unprivileged user using `sudo`.
2. Install the vFabric Suite 5.3 repository RPMs. These repository RPMs make it easy for you to browse all the available vFabric component RPMs. Use the following commands to install the vFabric repository RPM. The URLs differ depending on the version of RHEL you are using.

- For RHEL 5:

```
prompt# wget -q -O -
http://repo.vmware.com/pub/rhel5/vfabric/5.3/vfabric-5.3-suite-installer
| sh
```

- For RHEL 6:

```
prompt# wget -q -O -
http://repo.vmware.com/pub/rhel6/vfabric/5.3/vfabric-5.3-suite-installer
| sh
```

The command performs the following tasks:

- Imports the vFabric GNU Privacy Guard (GPG) key.
  - Installs the vFabric 5.3 repository RPM.
  - Launches the VMware End User License Agreement (EULA) acceptance and repository configuration script.
  - Outputs the EULA for you to read; you must answer yes to accept the terms and continue.
3. Execute the following `yum` command. If necessary, use `sudo` to run the command if you are not logged in as root:

```
prompt# yum install vfabric-gemfire
```

The `yum` command begins the install process, resolves dependencies, and displays the packages it plans to install.

The `yum install vfabric-gemfire` command installs the most recent version of the vFabric GemFire RPM that it finds in all installed repositories. If you want to install an earlier version of GemFire, you must explicitly specify the version/RPM name with the `yum install` command. For example:

```
prompt# yum install vfabric-gemfire-7.0.1-1
```

Use `yum search vfabric-gemfire --showduplicates` to find all versions that are available in the installed RPM repositories.

4. If this is the first time that you install a vFabric component on the VM, the `yum` command also installs the `vfabric-eula` RPM and prompts you to accept the VMware license agreement. You must answer **yes** to continue.
5. Enter **y** at the prompt to begin the actual installation.
6. If the installation is successful, you see a `Complete!` message at the end.

The `yum` command:

- Installs vFabric GemFire into the `/opt/vmware/vfabric_gemfire/` directory.
- If the user does not already exist, adds a `gemfire` non-interactive user (in a group called `vfabric`) and sets the owner of all directories and files under `/opt/vmware/vfabric_gemfire/` to `gemfire`.

Note that you cannot login directly as the `gemfire` user because interactive login has been disabled. Rather, you must login as the root user or as a privileged user using `sudo`, and then `su - gemfire`.

7. Configure your environment for GemFire.
- Set the `GEMFIRE` environment variable to point to your GemFire installation top-level directory. (You should see `bin`, `lib`, `dtd`, and other directories under `GEMFIRE`.)

```
GEMFIRE=/opt/vmware/vfabric-gemfire/vFabric_GemFire_701
export GEMFIRE
```

- Configure `GF_JAVA` and your `PATH` and `CLASSPATH` as shown in these examples. `GF_JAVA` must point to the `java` executable file under your `JAVA_HOME`. (If you have not done so already, you should also set your `JAVA_HOME` variable to a supported Java installation.)

If you will be using the `gfsh` command-line utility or managing servers and locators with the `ServerLauncher` and `LocatorLauncher` APIs, then you must set `JAVA_HOME` to a JDK installation and include `tools.jar` in your `CLASSPATH`. For example:

```
JAVA_HOME=/usr/java/jdk1.6.0_38
export JAVA_HOME
GF_JAVA=$JAVA_HOME/bin/java
export GF_JAVA
PATH=$PATH:$JAVA_HOME/bin:$GEMFIRE/bin
export PATH
CLASSPATH=$CLASSPATH:$GEMFIRE/lib/gemfire.jar:\
$GEMFIRE/lib/antlr.jar:\
$GEMFIRE/lib/gfsh-dependencies.jar:\
$GEMFIRE/lib/gfSecurityImpl.jar:\
$GEMFIRE/lib/jackson-core-asl-1.9.9.jar:\
$GEMFIRE/lib/commons-logging.jar:\
$GEMFIRE/lib/tomcat-embed-core.jar:\
$GEMFIRE/lib/tomcat-embed-logging-juli.jar:\
$GEMFIRE/lib/tomcat-embed-jasper.jar:\
$JAVA_HOME/lib/tools.jar
export CLASSPATH
```

8. If you have purchased a production license, activate the license. See [Understanding vFabric GemFire Licenses](#) on page 45 and [Installing and Configuring vFabric GemFire Licenses](#) on page 55.
9. Repeat this procedure for every virtual or physical machine on which you will run vFabric GemFire.

### What to do next

- If you obtained vFabric GemFire as a standalone product, install a license (serial number) on each physical and virtual machine on which you installed vFabric GemFire. See [Understanding vFabric GemFire Licenses](#) on page 45 and [Installing and Configuring vFabric GemFire Licenses](#) on page 55. (Note that you do not need to install any vFabric GemFire licenses on machines that are only running client applications.)
- If you obtained vFabric GemFire as part of GemFire, configure vFabric GemFire to communicate with the vFabric License Server. See [vFabric Suite Only: Configure GemFire for vFabric License Server](#) on page 58.
- Run the product tutorial and examples. See [Setting Up the Product Examples](#) on page 61 and [vFabric GemFire Tutorial](#) on page 65.
- If you need to uninstall GemFire, see [How to Uninstall GemFire](#) on page 34.

## Install vFabric GemFire from a ZIP File

For non-RHEL users, install and configure vFabric GemFire on every physical and virtual machine where you will run vFabric GemFire.

### Prerequisites

Before you install the vFabric GemFire, you must complete the following prerequisites:

- Confirm that your system meets the hardware and software requirements described in [Supported Configurations and System Requirements](#) on page 1.
- From the [VMware downloads page](#), select **VMware vFabric GemFire**. From the Download VMware vFabric GemFire 7.0 page, perform the following steps:
  - If you are installing GemFire for evaluation, click on the "**Download Free Trial**" link, register an account with VMware (if you have not done so already) and download GemFire.
  - If you have already purchased GemFire, locate the vFabric GemFire licensed offering (Data Management Node, Application Cache Node, Global WAN Upgrade or Unlimited Client Upgrade) that you have purchased from the Product Downloads tab. Click on the **View Download** button.



**Note:** You can also get GemFire from your salesperson.

- Download the ZIP distribution vFabric GemFire distributed as a ZIP file.
- Know how to configure environment variables for your system. If you have not done so already, set the JAVA\_HOME environment variable to point to a Java runtime installation supported by GemFire. (You should find a bin directory under JAVA\_HOME.)



**Note:** Certain features of the gfsh command-line utility require tools.jar from the JDK. Make sure that you set JAVA\_HOME to a JDK installation if you plan to use gfsh on the machine where you are installing GemFire.

### Procedure

Use the following procedure to install vFabric GemFire:

1. Navigate to the directory where you downloaded the GemFire software, and unzip the .zip file.

- **UNIX and Linux (Bourne and Korn shells - sh, ksh, bash).** If you are using the command line, type the following command:

```
$ unzip vFabric_GemFire_XXX_bNNNNN.zip -d path_to_product
```

where *XXX* corresponds to the product version of GemFire that you are installing, *bNNNNN* corresponds to the build number of the software, and *path\_to\_product* corresponds to the location where you want to install GemFire. For example:

```
$ unzip vFabric_GemFire_701_b40235.zip -d /opt/vmware/vfabric-gemfire
```

Alternatively, unzip the .zip file directly with any common ZIP extraction tool suitable for your operating system.

- **Windows:** Open the .zip file, and extract the product files to the desired installation location on your machine.

## 2. Configure your environment for GemFire.

- Set the GEMFIRE environment variable to point to your GemFire installation top-level directory. (You should see bin, lib, dtd, and other directories under GEMFIRE.) The following variables definitions are examples; your installation path may vary depending on where you install GemFire and the version you are installing.

- **UNIX and Linux (Bourne and Korn shells - sh, ksh, bash)**

```
GEMFIRE=/opt/vmware/vfabric-gemfire/vFabric_GemFire_701
export GEMFIRE
```

- **Windows**

```
set GEMFIRE=C:\vmware\vfabric-gemfire\vFabric_GemFire_701
```

- Configure your PATH, CLASSPATH and GF\_JAVA environment variables as shown in these examples. GF\_JAVA must point to the java executable file under your JAVA\_HOME. (If you have not done so already, you should also set your JAVA\_HOME variable to a supported Java installation.)

If you will be using the gfsh command-line utility or managing servers and locators with the ServerLauncher and LocatorLauncher APIs, then you must set JAVA\_HOME to a JDK installation and include tools.jar in your CLASSPATH. For example:

- **UNIX and Linux (Bourne and Korn shells - sh, ksh, bash)**

```
JAVA_HOME=/usr/java/jdk1.6.0_38
export JAVA_HOME
GF_JAVA=$JAVA_HOME/bin/java
export GF_JAVA
PATH=$PATH:$JAVA_HOME/bin:$GEMFIRE/bin
export PATH
CLASSPATH=$CLASSPATH:$GEMFIRE/lib/gemfire.jar:\
$GEMFIRE/lib/antlr.jar:\
$GEMFIRE/lib/gfsh-dependencies.jar:\
$GEMFIRE/lib/gfSecurityImpl.jar:\
$GEMFIRE/lib/jackson-core-asl-1.9.9.jar:\
$GEMFIRE/lib/commons-logging.jar:\
$GEMFIRE/lib/tomcat-embed-core.jar:\
$GEMFIRE/lib/tomcat-embed-logging-juli.jar:\
$GEMFIRE/lib/tomcat-embed-jasper.jar:\
$JAVA_HOME/lib/tools.jar
export CLASSPATH
```

- **Windows**

```
set JAVA_HOME=c:\Program Files\Java\jdk1.7.0_02
set GF_JAVA=%JAVA_HOME%\bin\java.exe
```

```
set PATH=%PATH%;%JAVA_HOME%\bin;%GEMFIRE%\bin
set CLASSPATH=%CLASSPATH%^
%GEMFIRE%\lib\gemfire.jar;^
%GEMFIRE%\lib\antlr.jar;^
%GEMFIRE%\lib\gfSecurityImpl.jar;^
%GEMFIRE%\lib\jackson-core-asl-1.9.9.jar;^
%GEMFIRE%\lib\commons-logging.jar;^
%GEMFIRE%\lib\tomcat-embed-core.jar;^
%GEMFIRE%\lib\tomcat-embed-logging-juli.jar;^
%GEMFIRE%\lib\tomcat-embed-jasper.jar;^
%JAVA_HOME%\lib\tools.jar
```



**Note:** You only need to add `commons-logging.jar`, `tomcat-embed-core.jar`, `tomcat-embed-logging-juli.jar`, and `tomcat-embed-jasper.jar` files to your `CLASSPATH` if you are planning on using the GemFire member as a GemFire Pulse application server.

3. If you have purchased a production license, activate the license. See [Understanding vFabric GemFire Licenses](#) on page 45 and [Installing and Configuring vFabric GemFire Licenses](#) on page 55.
4. Repeat this procedure for every virtual or physical machine on which you will run vFabric GemFire.

### Obtaining Modified Open Source Code Libraries

Many open source licenses require that vendors who use or modify their libraries make that code available.

To obtain the open source code libraries modified in GemFire, visit the product download page, and select the **Open Source** tab. Download the `vFabric_GemFire_XXX_bNNNNN_OSS.zip` file (where `XXX` corresponds to the product version of GemFire that you are installing and `bNNNNN` corresponds to the build number of the software.) Download and read the associated disclosure information (`.txt`) file from the same download page.

The `.zip` file contains both the original open source libraries and the modified source libraries.

### What to Do Next

After you install vFabric GemFire from a `.zip` file, you perform the following tasks:

- If you obtained vFabric GemFire as a standalone product, install a license (serial number) on each physical and virtual machine on which you installed vFabric GemFire. See [Understanding vFabric GemFire Licenses](#) on page 45 and [Installing and Configuring vFabric GemFire Licenses](#) on page 55.
- Run the product tutorial and examples. See [Setting Up the Product Examples](#) on page 61 and [vFabric GemFire Tutorial](#) on page 65.
- If you need to uninstall GemFire, see [How to Uninstall GemFire](#) on page 34.

### How to Uninstall GemFire

This section describes how to remove GemFire.

If you installed vFabric GemFire from a ZIP file, shut down any running GemFire processes and then simply delete the product tree to uninstall the product. No additional registry entries or system modifications are needed.

If you are using RHEL and installed vFabric GemFire using RPMs, you can use the `rpm -e` command to uninstall GemFire. For example:

```
prompt# sudo -u gemfire -E rpm -e vfabric-gemfire
```

# Chapter 9

---

## Upgrading vFabric GemFire

To upgrade vFabric GemFire from an existing installation, you install the latest version of the software from a JAR file, or if your system is RHEL, by using RPMs. An upgrade from one major version to another (for example, from GemFire 6.6.x to 7.0.x) may require brief downtime for your system.

### Before You Upgrade

This section provides information you need to know before you begin to upgrade vFabric GemFire.

Before you upgrade, review the following items:

- During an upgrade from one major GemFire version to another, you cannot connect members in the same distributed system to other members running a different major version. Therefore, you cannot perform a rolling upgrade between major versions of vFabric GemFire and will need to plan your system downtime accordingly. See [vFabric GemFire Version Compatibility Rules](#) on page 44 for more information on GemFire version interoperability.
- Read the vFabric GemFire Release Notes at <https://www.vmware.com/support/pubs/vfabric-gemfire.html> to familiarize yourself with all feature changes.
- Know how to configure environment variables for your system.
- Confirm that your system meets the requirements to run GemFire. See [Supported Configurations and System Requirements](#) on page 1.



**Note:** To check your current Java version, type `java -version` at a command-line prompt. You can download Sun/Oracle Java SE from the following location:  
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

- If you are upgrading to GemFire 7.0.x from GemFire version 6.5.x or earlier, you will need first to upgrade your disk store files by using the DiskConverter tool. See [DiskConverterPre65to65](#) on page 817 for instructions on running the tool.
- If you are upgrading from GemFire 6.6 (or have just upgraded your disks to 6.6.x compatibility with the DiskConverter tool), you must run the `gfs` command `upgrade offline-disk-store` tool to upgrade the disk stores to vFabric Gemfire 7.0.x.
- If you obtain GemFire as part of a vFabric Suite package, first complete the vFabric License Server installation and license activation procedures in [Getting Started with vFabric Suite](#). Then follow procedures in this document to set up your environment for GemFire and to complete any remaining GemFire-specific installation and configuration tasks.

## RHEL: Upgrade vFabric GemFire (Standalone) from RPM

If your guest operating system is Red Hat Enterprise Linux (RHEL) and you have installed a previous version of standalone GemFire using RPM, VMware recommends that you use RPM distribution to upgrade vFabric GemFire. You complete the upgrade procedure on every virtual and physical machine that will run GemFire.



**Note:** Thoroughly test your development systems with the new version before moving into production.

When you upgrade vFabric GemFire on RHEL from RPM, the software is installed by default into `/opt/vmware/vfabric-gemfire/vFabric_GemFire_XXX` where XXX corresponds to the version of GemFire (for example, `vFabric_GemFire_701`) that you have installed. The vFabric GemFire installation is owned by the `gemfire` user in the `vfabric` group.

When you upgrade vFabric GemFire using the RPM, the upgrade process installs the new version into its own installation directory (`/opt/vmware/vfabric_gemfire/vFabric_GemFire_XXX` where XXX corresponds to the newly installed version). No files are overwritten during the upgrade process. Therefore, you can have two different versions of GemFire installed on the same machine for the purposes of performing a rolling upgrade. See [Performing a Rolling Upgrade](#) on page 43 for more information.

### Prerequisites

Before you upgrade, complete the following prerequisites:

- Confirm that your system meets the hardware and software requirements described in [Supported Configurations and System Requirements](#) on page 1.
- Read [Before You Upgrade](#) on page 35.
- Use the following steps to download the vFabric GemFire RPM.
  - From the [VMware downloads page](#), select **VMware vFabric GemFire**.
  - If you are installing GemFire for evaluation, click "Try Now" button to download an evaluation RPM.
  - If you have purchased GemFire, download the vFabric GemFire offering you have purchased from the Product Downloads tab. You can also get GemFire from your salesperson.
  - Download the vFabric GemFire RPM appropriate for your RHEL operating system.
    - RHEL 5: `vfabric-gemfire-7.0.1-2.el5.noarch.rpm`
    - RHEL 6: `vfabric-gemfire-7.0.1-2.el6.noarch.rpm`

### Procedure for Upgrading vFabric GemFire 6.x.x to 7.0.x

Use the following procedure to upgrade vFabric GemFire 6.x.x to 7.0.x using RPM:

1. Review the items listed in [Prerequisites](#) on page 39 and make any appropriate preparations.
2. Execute the following `rpm` command. If necessary, use `sudo` to run the command either as root or as the `gemfire` user. For example:

#### RHEL 5:

```
prompt# sudo -u gemfire -E rpm -ivh vfabric-gemfire-7.0.1-2.el5.noarch.rpm
```

#### RHEL 6:

```
prompt# sudo -u gemfire -E rpm -ivh vfabric-gemfire-7.0.1-2.el6.noarch.rpm
```

The `rpm` command begins the install process, resolves dependencies, and displays the packages it plans to install.

3. Repeat this upgrade procedure for every virtual or physical machine on which you will run vFabric GemFire.
4. Recompile your Java applications against the `gemfire.jar` in the 7.0.X version of the product.

5. Stop all members of the system running with the prior 6.x.x version.

- a. Shut down all members running a cache by using the `gemfire shut-down-all` command:

```
sudo -u gemfire -E gemfire -J-DgemfirePropertyFile=mygemfire.properties
shut-down-all
```

In the sample command, substitute *mygemfire.properties* with the location of the previous vFabric GemFire's *gemfire.properties* file for the distributed system you are shutting down.

- b. Shut down any locators. To shut down a locator in 6.x.x deployments, issue the following command:

```
gemfire stop-locator -port=port -address=ipAddr -dir=locatorDir
```

Replace *port*, *ipAddr* and *locatorDir* with the appropriate values.

See [Shutting Down Your System](#) in the vFabric GemFire 6.6 documentation set for more details.

6. Upgrade your disk store files.

- If you will be upgrading to GemFire 7.0.x from a GemFire version earlier than 6.5.x, upgrade your disk store files by using the `DiskConverterPre65to65` tool. See [DiskConverterPre65to65](#) on page 817 for instructions on running the tool.
- If you are upgrading from GemFire 6.5.x or 6.6.x (or have just upgraded your disks to 6.5 compatibility with the `DiskConverterPre65to65` tool), run the `gfsh` command `upgrade offline-disk-store` tool to upgrade the disk stores to vFabric Gemfire 7.0.x. See [upgrade offline-disk-store](#) on page 732 for more information.

7. Redeploy your environment's license or configuration files to the new version's installation. For example, you may need to do one of the following tasks depending on your GemFire deployment configuration:

- If you are upgrading from GemFire 6.6.x or later, you may need to copy any existing license files to the new product installation location. See [How GemFire Manages Licenses](#) on page 52 for information for possible license file locations.
- If you are using common configuration files, update your configuration files as required with the path to the new installation. See [Deploying vFabric GemFire Configuration Files](#) on page 631 for more information on how to deploy configuration files.
- If you are using local configuration files, update each set of configuration files that are local to the distributed system you are upgrading. If you are using local files, you also have the option to do a rolling upgrade and upgrade each distributed system one at a time. See [Deploying vFabric GemFire Configuration Files](#) on page 631 for more information on how to deploy configuration files.

8. Point all member sessions to the new installation of GemFire. For example, you may need to do one of the following tasks depending on your application's configuration:

- Modify your applications to point to the new GemFire product tree location.
- Copy the `gemfire.jar` file out of the new GemFire product tree location and replace the existing `gemfire.jar` file in your application.

9. Restart all system members according to your usual procedures. See [Starting Up Your System](#) on page 637 for more information.

## Procedure for Upgrading vFabric GemFire 7.0.0 to 7.0.x

The following procedure describes how to upgrade your entire distributed system from vFabric GemFire 7.0.0 to 7.0.x at once. You also have the option of performing a rolling upgrade. See [Performing a Rolling Upgrade](#) on page 43.

1. Review the items listed in [Prerequisites](#) on page 39 and make any appropriate preparations.
2. Execute the following `rpm` command. If necessary, use `sudo` to run the command either as root or as the `gemfire` user. For example:

**RHEL 5:**

```
prompt# sudo -u gemfire -E rpm -ivh vfabric-gemfire-7.0.1-2.el5.noarch.rpm
```

**RHEL 6:**

```
prompt# sudo -u gemfire -E rpm -ivh vfabric-gemfire-7.0.1-2.el6.noarch.rpm
```

The `rpm` command begins the install process, resolves dependencies, and displays the packages it plans to install.

3. Repeat this upgrade procedure for every virtual or physical machine on which you will run vFabric GemFire.
4. Stop all members of the system running with the prior version. For example:

- a. Open a gfsh prompt.

- b. Make sure you are connected (via its JMX Manager) to the distributed system you wish to shutdown.

- c. At the gfsh prompt, type:

```
gfsh>shutdown
```

- d. Type **Y** at the prompt to shut down your entire distributed system.

- e. Shut down each locator in the distributed system by using the `stop locator` command. For example:

```
gfsh>stop locator --name=locator_name
```

See [Shutting Down the System](#) on page 638 for more details.

5. Point all member sessions to the new installation of GemFire. For example, you may need to do one of the following tasks depending on your application's configuration:
  - Modify your applications to point to the new GemFire product tree location.
  - Copy the `gemfire.jar` file out of the new GemFire product tree location and replace the existing `gemfire.jar` file in your application.
6. Restart all system members according to your usual procedures. See [Starting Up Your System](#) on page 637 for more information.

## What to Do Next

After you upgrade vFabric GemFire from RPM, you can perform the following tasks:

- Run the product tutorial and examples. See [Setting Up the Product Examples](#) on page 61 and [vFabric GemFire Tutorial](#) on page 65.
- Test your development systems with the new version.
- If you need to uninstall GemFire, see [How to Uninstall GemFire](#) on page 34.

## RHEL: Upgrade vFabric GemFire (vFabric Suite) from the VMware YUM Repository

If your guest operating system is Red Hat Enterprise Linux (RHEL) and you have installed a previous version of GemFire using yum and the VMware RPM repository, VMware recommends that you use yum to upgrade vFabric GemFire. You complete the upgrade procedure on every virtual and physical machine that will run GemFire.



**Note:** Thoroughly test your development systems with the new version before moving into production.

When you upgrade vFabric Suite components on RHEL from the VMware RPM repository, the different suite components are installed into different directories and are owned by different users in different groups. The vFabric GemFire installation is owned by the `gemfire` user in the `vfabric` group.

The default installation directory for GemFire is

`/opt/vmware/vfabric_gemfire/vFabric_GemFire_XXX` where XXX corresponds to the version of GemFire (for example, vFabric\_GemFire\_701) that you have installed.

When you upgrade vFabric GemFire using the RPM, the upgrade process installs the new version into its own installation directory (`/opt/vmware/vfabric_gemfire/vFabric_GemFire_XXX` where XXX corresponds to the newly installed version). No files are overwritten during the upgrade process. Therefore, you can have two different versions of GemFire installed on the same machine for the purposes of performing a rolling upgrade.

## Prerequisites

Before you upgrade, complete the following prerequisites:

- Confirm that your system meets the hardware and software requirements described in [Supported Configurations and System Requirements](#) on page 1.
- Read [Before You Upgrade](#) on page 35.
- Install the vFabric 5 RPMs (you do this procedure only once):

1. Log in to the RHEL VM as the root user (or as an unprivileged user who has sudo privileges) and start a terminal.
2. Install the vFabric Suite 5.3 repository RPMs. These repository RPMs make it easy for you to browse all the available vFabric component RPMs. Use the following commands to install the vFabric repository RPM. The URLs differ depending on the version of RHEL you are using.

- For RHEL 5:

```
prompt# wget -q -O -
http://repo.vmware.com/pub/rhel5/vfabric/5.3/vfabric-5.3-suite-installer
| sh
```

- For RHEL 6:

```
prompt# wget -q -O -
http://repo.vmware.com/pub/rhel6/vfabric/5.3/vfabric-5.3-suite-installer
| sh
```

The command performs the following tasks:

- Imports the vFabric GNU Privacy Guard (GPG) key.
  - Installs the vFabric 5.3 repository RPM.
  - Launches the VMware End User License Agreement (EULA) acceptance and repository configuration script.
  - Outputs the EULA for you to read; you must answer yes to accept the terms and continue.
3. Use the `yum search vfabric` command to view the list of vFabric components that you can install from the VMware repository. The vFabric GemFire RPM is called `vfabric-gemfire`.

## Procedure for Upgrading vFabric GemFire 6.x.x to 7.0.x

Use the following procedure to upgrade vFabric GemFire 6.x.x to 7.0.x using RPM:

1. Review the items listed in [Prerequisites](#) on page 39 and make any appropriate preparations.
2. From the RHEL computer on which you will upgrade vFabric GemFire, log in as the root user or as an unprivileged user using sudo.
3. Execute the following yum command:

```
prompt# yum upgrade vfabric-gemfire
```

The `yum` command begins the upgrade process, resolves dependencies, and displays the packages it plans to install.

The `yum upgrade vfabric-gemfire` command upgrades to the most recent version of the vFabric GemFire RPM that it finds in all installed repositories. If you want to upgrade to a different version of GemFire, you must explicitly specify the version with the `yum install` command. For example:

```
prompt# yum upgrade vfabric-gemfire-701
```

Use `yum search vfabric-gemfire --showduplicates` to find all versions that are available in the installed RPM repositories.

4. Enter `y` at the prompt to begin the actual upgrade. If you have not already accepted the VMware license terms, a prompt asks you and you must answer yes to continue.  
If the upgrade is successful, you see a `Complete!` message at the end.
5. Repeat this upgrade procedure for every virtual or physical machine on which you will run vFabric GemFire.
6. Recompile your Java applications against the `gemfire.jar` in the 7.0.X version of the product.
7. Stop all members of the system running with the prior 6.x.x version.

- a. Shut down all members running a cache by using the `gemfire shut-down-all` command:

```
sudo -u gemfire -E gemfire -J-DgemfirePropertyFile=mygemfire.properties  
shut-down-all
```

In the sample command, substitute `mygemfire.properties` with the location of the previous vFabric GemFire's `gemfire.properties` file for the distributed system you are shutting down.

- b. Shut down any locators. To shut down a locator in 6.x.x deployments, issue the following command:

```
gemfire stop-locator -port=port -address=ipAddr -dir=locatorDir
```

Replace `port`, `ipAddr` and `locatorDir` with the appropriate values.

See [Shutting Down Your System](#) in the vFabric GemFire 6.6 documentation set for more details.

8. Upgrade your disk store files.
  - If you will be upgrading to GemFire 7.0.x from a GemFire version earlier than 6.5.x, upgrade your disk store files by using the `DiskConverterPre65to65` tool. See [DiskConverterPre65to65](#) on page 817 for instructions on running the tool.
  - If you are upgrading from GemFire 6.5.x or 6.6.x (or have just upgraded your disks to 6.5 compatibility with the `DiskConverterPre65to65` tool), run the `gfsh` command `upgrade offline-disk-store` tool to upgrade the disk stores to vFabric Gemfire 7.0.x. See [upgrade offline-disk-store](#) on page 732 for more information.
9. Redeploy your environment's license or configuration files to the new version's installation. For example, you may need to do one of the following tasks depending on your GemFire deployment configuration:
  - If you are upgrading from GemFire 6.6 or later, you may need to copy any existing license files to the new product installation location. See [How GemFire Manages Licenses](#) on page 52 for information for possible license file locations.
  - If you are using common configuration files, update your configuration files as required with the path to the new installation. See [Deploying vFabric GemFire Configuration Files](#) on page 631 for more information on how to deploy configuration files.
  - If you are using local configuration files, update each set of configuration files that are local to the distributed system you are upgrading. If you are using local files, you also have the option to do a rolling upgrade and upgrade each distributed system one at a time. See [Deploying vFabric GemFire Configuration Files](#) on page 631 for more information on how to deploy configuration files.
10. Point all member sessions to the new installation of GemFire. For example, you may need to do one of the following tasks depending on your application's configuration:

- Modify your applications to point to the new GemFire product tree location.
  - Copy the `gemfire.jar` file out of the new GemFire product tree location and replace the existing `gemfire.jar` file in your application.
11. Restart all system members according to your usual procedures. See [Starting Up Your System](#) on page 637 for more information.

## What to Do Next

After you upgrade vFabric GemFire from RPM, you can perform the following tasks:

- Run the product tutorial and examples. See [Setting Up the Product Examples](#) on page 61 and [vFabric GemFire Tutorial](#) on page 65.
- Test your development systems with the new version.
- If you need to uninstall GemFire, see [How to Uninstall GemFire](#) on page 34.

## Upgrade vFabric GemFire from a ZIP File

For non-RHEL users, upgrade vFabric GemFire on every virtual and physical machine that will run GemFire.

### Procedure for Upgrading vFabric GemFire 6.X to 7.0.X

Use the following procedure to upgrade your vFabric GemFire deployment:



**Note:** Thoroughly test your development systems with the new version before moving into production.

1. Review the items listed in [Before You Upgrade](#) on page 35 and make any appropriate preparations.
  2. Install the latest version of GemFire in a *different directory* than the existing version. See [Install vFabric GemFire from a ZIP File](#) on page 32.
- 
- Note:** GemFire is installed as a complete product, rather than as a modification to a prior version. In addition, the Java JRE runtime environment is not bundled with GemFire, so you need to have an appropriate JDK or JRE installed and configured to comply with GemFire requirements and your unique system needs. See [Supported Configurations and System Requirements](#) on page 1 for the configuration requirements of GemFire 7.0.x.
3. Review the changes and upgrade notes that are documented in the vFabric GemFire Release Notes.
    - Review the vFabric GemFire 7.0.x Release Notes that are available at <https://www.vmware.com/support/pubs/vfabric-gemfire.html>. Make any changes to your applications that are required for you to migrate to this version.
    - If you are upgrading to 7.0.x from a version earlier than GemFire 6.6, review all changes documented in the vFabric GemFire 6.6.x Release Notes. (At <https://www.vmware.com/support/pubs/vfabric-gemfire.html>, select Release 6.6.x.) Make any changes to your programs required for you to migrate to this version. In particular, you must update your licensing and licensing configuration to upgrade to GemFire 6.6.x.
  4. Recompile your Java applications against the `gemfire.jar` in this version of the product. After you have recompiled your Java applications, the location where you place the updated JARs will vary depending on your configuration. You may put them in the GemFire product tree or you could directly copy in the `gemfire.jar` file to a location appropriate for your application.
  5. Stop all members of the system running with the prior 6.x.x version.
    - a. Shut down all members running a cache by using the `gemfire shut-down-all` command:  

```
sudo -u gemfire -E gemfire -J-DgemfirePropertyFile=mygemfire.properties
shut-down-all
```

In the sample command, substitute *mygemfire.properties* with the location of the previous vFabric GemFire's *gemfire.properties* file for the distributed system you are shutting down.

- b. Shut down any locators. To shut down a locator in 6.x.x deployments, issue the following command:

```
gemfire stop-locator -port=port -address=ipAddr -dir=locatorDir
```

Replace *port*, *ipAddr* and *locatorDir* with the appropriate values.

See [Shutting Down Your System](#) in the vFabric GemFire 6.6 documentation set for more details.

6. Upgrade your disk store files.
  - If you will be upgrading to GemFire 7.0.x from a GemFire version earlier than 6.5.x, upgrade your disk store files by using the *DiskConverterPre65to65* tool. See [DiskConverterPre65to65](#) on page 817 for instructions on running the tool.
  - If you are upgrading from GemFire 6.5.x or 6.6.x (or have just upgraded your disks to 6.5 compatibility with the *DiskConverterPre65to65* tool), run the *gfsh* command *upgrade offline-disk-store* tool to upgrade the disk stores to vFabric Gemfire 7.0.x. See [upgrade offline-disk-store](#) on page 732 for more information.
7. Redeploy your environment's license or configuration files to the new version's installation. For example, you may need to do one of the following tasks depending on your GemFire deployment configuration:
  - If you are upgrading from GemFire 6.6.x or later, you may need to copy any existing license files to the new product installation location. See [How GemFire Manages Licenses](#) on page 52 for information for possible license file locations.
  - If you are using common configuration files, update your configuration files as required with the path to the new installation. See [Deploying vFabric GemFire Configuration Files](#) on page 631 for more information on how to deploy configuration files.
  - If you are using local configuration files, update each set of configuration files that are local to the distributed system you are upgrading. If you are using local files, you also have the option to do a rolling upgrade and upgrade each distributed system one at a time. See [Deploying vFabric GemFire Configuration Files](#) on page 631 for more information on how to deploy configuration files.
8. Point all member sessions to the new installation of GemFire. For example, you may need to do one of the following tasks depending on your application's configuration:
  - Modify your applications to point to the new GemFire product tree location.
  - Copy the *gemfire.jar* file out of the new GemFire product tree location and replace the existing *gemfire.jar* file in your application.
9. Restart all system members according to your usual procedures. See [Starting Up Your System](#) on page 637 for more information.

### **Procedure for Upgrading vFabric GemFire 7.0.0 to 7.0.x**

The following procedure describes how to upgrade your entire distributed system from vFabric GemFire 7.0.0 to 7.0.x at once. You also have the option of performing a Rolling Upgrade. See [Performing a Rolling Upgrade](#) on page 43.

1. Review the items listed in [Before You Upgrade](#) on page 35 and make any appropriate preparations.
2. Install the latest version of GemFire in a *different directory* than the existing version. See [Install vFabric GemFire from a ZIP File](#) on page 32.



**Note:** GemFire is installed as a complete product, rather than as a modification to a prior version. The Java JRE runtime environment is not bundled with GemFire, so you need to have an appropriate JDK or JRE installed and configured to comply with GemFire requirements and your unique system needs.

3. Repeat this upgrade procedure for every virtual or physical machine on which you will run vFabric GemFire.
4. Stop all members of the system running with the prior version. For example, using the `gfsh shutdown` command:
  - a. Open a gfsh prompt.
  - b. Make sure you are connected (via its JMX Manager) to the distributed system you wish to shutdown.
  - c. At the gfsh prompt, type:  
`gfsh>shutdown`
  - d. Type `y` at the prompt to shutdown your entire distributed system.
  - e. Shut down each locator in the distributed system by using the `stop locator` command. For example:  
`gfsh>stop locator --name=locator_name`

See [Shutting Down the System](#) on page 638 for more details.

5. Point all member sessions to the new installation of GemFire. For example, you may need to do one of the following tasks depending on your application's configuration:
  - Modify your applications to point to the new GemFire product tree location.
  - Copy the `gemfire.jar` file out of the new GemFire product tree location and replace the existing `gemfire.jar` file in your application.
6. Restart all system members according to your usual procedures. See [Starting Up Your System](#) on page 637 for more information.

## What to Do Next

After you upgrade, perform the following tasks:

- Run the product tutorial and examples. See [Setting Up the Product Examples](#) on page 61 and [vFabric GemFire Tutorial](#) on page 65.
- Test your development systems with the new version.
- If you are upgrading from a version earlier than GemFire 6.6.0, set up licensing. See [Installing and Configuring vFabric GemFire Licenses](#) on page 55.
- If you need to uninstall GemFire, see [How to Uninstall GemFire](#) on page 34.

## Performing a Rolling Upgrade

A rolling upgrade allows you to keep your existing distributed system running while you upgrade your members gradually.

### Supported Versions for Rolling Upgrade

GemFire supports a rolling upgrade between minor versions. A minor version corresponds to an increment in the third place position in a version number. For example, you can perform a rolling upgrade from vFabric GemFire 6.6.1 to 6.6.4 or GemFire 7.0.0 to 7.0.1.

A rolling upgrade is not supported between major versions. A major version corresponds to an increment in either the second or the first position in a version number. For example, you cannot perform a rolling upgrade from vFabric GemFire 6.5 to 6.6 or a rolling upgrade from GemFire 6.6.3 to 7.0.1.



**Note:** Rolling upgrade applies only to the peer members or cache servers within a distributed system cluster. It does not apply to overall multi-site (WAN) or client-server deployments since the version interoperability is looser between clients and servers and between different WAN sites. See [vFabric GemFire Version Compatibility Rules](#) on page 44 for more details on how different versions of GemFire can interoperate.

## How to Perform a Rolling Upgrade

This section describes the process used to perform a rolling upgrade.

### To upgrade each member process, perform the following steps:

1. Install the new version of the software (alongside the older version of the software) on the node (either via ZIP or RPM).
2. Stop the process that you are upgrading.
3. Point the upgraded member to the new installation of GemFire and update the environment. For example, you may need to do one or more of the following tasks depending on your application's configuration:
  - a. Update your classpath.
  - b. Modify your applications to point to the new GemFire product tree location.
  - c. Copy the gemfire.jar file out of the new GemFire product tree location and replace the existing gemfire.jar file in your application.
4. Restart your process.
5. Check for successful upgrade on the member.
6. Upgrade the next member.

## vFabric GemFire Version Compatibility Rules

This topic describes the compatibility rules for deploying different versions of vFabric GemFire.

### Version Compatibility Between Peers and Cache Servers

You can have two different minor versions of GemFire running in the same distributed system cluster. For example, you can have peers or cache servers running vFabric GemFire 7.0.0 and 7.0.1 at the same time for the purposes of a rolling upgrade. You cannot run different major versions in the same distributed system cluster. See [Performing a Rolling Upgrade](#) on page 43 for more details on how to upgrade your system gradually between minor versions.

### Version Compatibility Between Clients and Servers

GemFire clients can run an older version of GemFire and still connect to GemFire servers running a newer version. In other words, a client running GemFire 5.7 will be able to connect to a server running GemFire 6.6.2 or GemFire 7.0.1. Newer clients such as GemFire 7.0.1 clients, however, cannot connect to servers running older versions of GemFire such as GemFire 6.6.4.

### Version Compatibility Between Sites in Multi-Site (WAN) Deployments

In multi-site (WAN) deployments, you can run different major versions of vFabric GemFire on the different sites. For example, one site can be running GemFire 5.7 and another site running GemFire 6.6.2 and the sites should still be able to communicate with one another.

# Chapter 10

---

## Understanding vFabric GemFire Licenses

You can choose among different types and quantities of vFabric GemFire production licenses, depending on your deployment topology. GemFire also includes a default evaluation license that enables you to run the product examples and perform simple evaluation activities.

### vFabric Suite and vFabric GemFire Licenses

How you install and consume your license depends on whether you obtain vFabric GemFire as a standalone product or as part of vFabric Suite (Standard or Advanced).

#### vFabric Suite Licenses

The default vFabric GemFire license offering when you purchase vFabric Suite (Standard or Advanced) is the Application Cache Node license. However, you can also purchase Data Management Node licenses if wish to deploy a client-server or WAN deployment.

To understand how vFabric Suite licensing works and to activate a vFabric Suite license, refer to [Getting Started with vFabric Suite](#).

#### vFabric GemFire Licenses and Serial Numbers

When you purchase vFabric GemFire as a standalone product, you get one or more serial numbers to use in your vFabric GemFire member configurations. You typically install the serial numbers locally, on every virtual and physical machine that will run vFabric GemFire.



**Note:** You do not need licenses for GemFire locators or for machines that are only used to run gfsh interface to connect to a vFabric GemFire distributed system. You also do not need to install serial numbers on GemFire clients.

Every Application Cache Node and Data Management Node serial number has a specific quantity of licenses embedded within it. An individual license is used for each 6-core CPU; license quantity reflects the number of actual 6-core CPUs that are allowed to run either GemFire clients or GemFire peer nodes. Therefore, if you are running multiple GemFire processes on a single 6-core CPU, you are still only consuming one license.

You receive a separate serial number for each license type (Application Cache Node and Data Management Node) and for each upgrade that you buy.

To purchase GemFire licenses, see <http://www.vmware.com/products/vfabric-gemfire/buy.html> or contact your VMware account manager.

## vFabric GemFire License Options

For production licensing, vFabric GemFire offers Application Cache Node and Data Management Node options. The Data Management Node license includes two upgrade options. You can combine these licenses, depending on your topology requirements. For evaluation licensing, a default evaluation license is included in the product, and you can request a custom evaluation license.

### Application Cache Node

Application Cache Node provides licensing for peer-to-peer systems. It is also used in conjunction with a Data Management Node license in client/server deployments to license a specific number of clients.

**Note to vFabric Suite Customers:** A vFabric Suite license includes vFabric GemFire Application Cache Node. If you are a vFabric Suite customer and want to deploy a client/server system or a WAN, you need to add Data Management Node licensing. Upgrades to unlimited clients and global WAN are available with DMN. You can install DMN licensing (serial numbers) locally, or through the vFabric License Server, as you did with vFabric Application Cache Node and other vFabric Suite products. If you used the License Server, refer to [vFabric Suite Only: Configure GemFire for vFabric License Server](#) on page 58.

### Data Management Node

A Data Management Node license is required for servers in client/server deployments. You can also install Unlimited Client Upgrade and Global WAN Upgrade options alongside this license. Unlimited Client Upgrade and Global WAN Upgrade can only be used in conjunction with the Data Management Node.

### Evaluation Licenses

The default evaluation license included with GemFire allows you to run up to three peer or cache server members and up to three clients for each cache server member. GemFire uses the default evaluation license upon startup if custom evaluation or production licensing has not been installed in GemFire. This license never expires. It is not allowed for production use. You do not need to install the default evaluation license; it activates automatically if you have not configured specific licensing through serial numbers or dynamic licensing.

You can also obtain custom evaluation licenses for your specific evaluation requirements. Custom evaluation license options are similar to those for [Application Cache Node](#) on page 46, but a custom evaluation license expires, while a production license does not. Contact your VMware account manager for details.

## Choosing a License Option Based on Topology

How you plan to use vFabric GemFire determines which GemFire license offerings are right for your system. Licensing requirements for vFabric GemFire are based on your deployment topology.

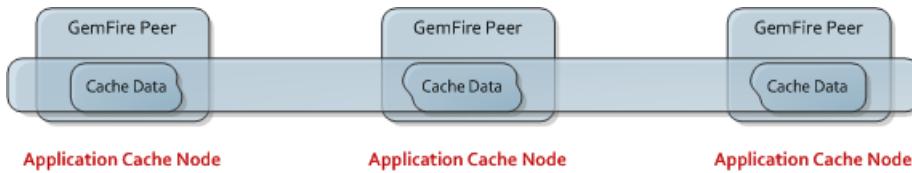
Licensing options are available for the following topologies.

- [Peer-to-Peer](#) on page 47
- [Peer-to-Peer with Global WAN](#) on page 47
- [Client/Server with a Limited Number of Clients](#) on page 48
- [Client/Server with an Unlimited Number of Clients](#) on page 49
- [Client/Server with Global WAN and a Limited Number of Clients](#) on page 50
- [Client/Server with Global WAN and Unlimited Clients](#) on page 51

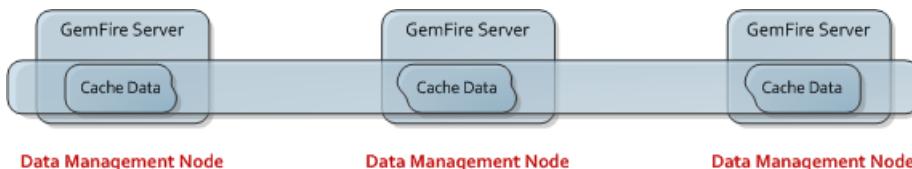
For detailed information about topology concepts and the topology options described in the table, see [Topologies and Communication](#) on page 161.

## Peer-to-Peer

Typically a simple peer-to-peer system requires an Application Cache Node serial number installed in each peer member.



You can license a peer-to-peer system of server peers using a Data Management Node license, or you can combine Application Cache Node peers and Data Management Node server peers in the same distributed system depending on your WAN and client needs.



Note that the Data Management Node license with Global WAN Upgrade is required if you need to connect caches through a WAN. See [Peer-to-Peer with Global WAN](#) on page 47 for complete details.

To install licensing for either of these topologies, perform the following steps:

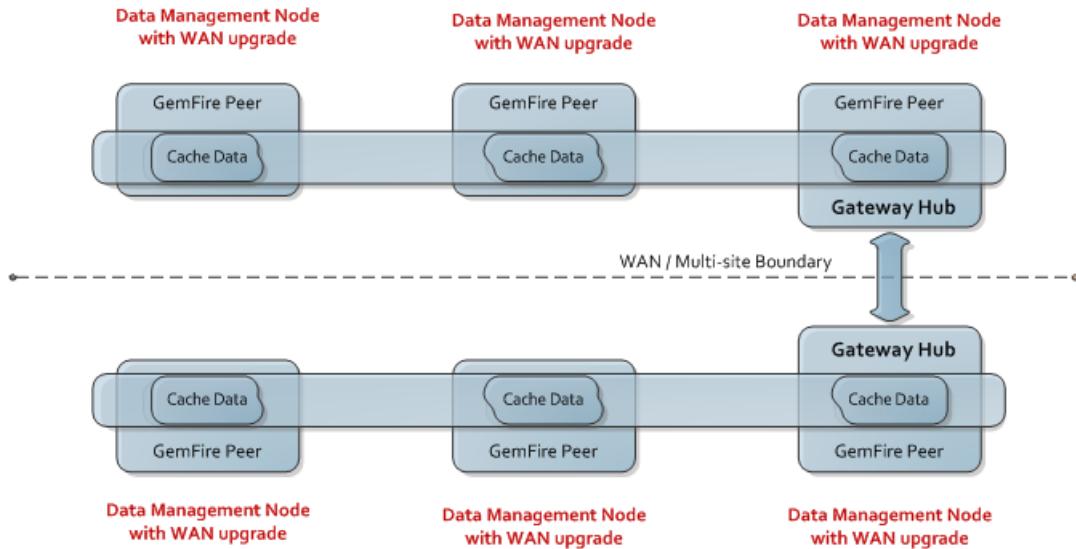
1. Obtain an Application Cache Node or a Data Management Node license from VMware.
2. Install the license in GemFire by using one of the following options:
  - Add either the Application Cache Node license serial number or the Data Management Node license serial number directly in `gemfire.properties` on each peer or server member. See [Option 1: Install Serial Numbers in gemfire.properties](#) on page 55; or
  - Create a file that contains the Application Cache Node license serial number or create a file that contains the Data Management Node license serial number. Copy the file to the vFabric serial number directory on all peer or server members. See [Option 2: Install Serial Numbers in vFabric Serial Number Files](#) on page 57; or
  - Install the Application Cache Node license or the Data Management Node license using vFabric License Server. See [vFabric Suite Only: Configure GemFire for vFabric License Server](#) on page 58.

## Peer-to-Peer with Global WAN

A peer-to-peer WAN system requires a Data Management Node license (serial number) with a Global WAN upgrade serial number. The members can be servers or simply peers that share data but do not service client connections. Each distributed system that participates in the WAN installation requires the configuration of a Data Management Node serial number and a Global WAN Upgrade serial number.



**Note:** You can use the same set of serial numbers on all distributed systems in a WAN deployment scenario; however, VMware recommends that you obtain a set of unique serial numbers for each distributed system that connects through the WAN. Each member of each distributed system communicating through the WAN must be installed with a license.



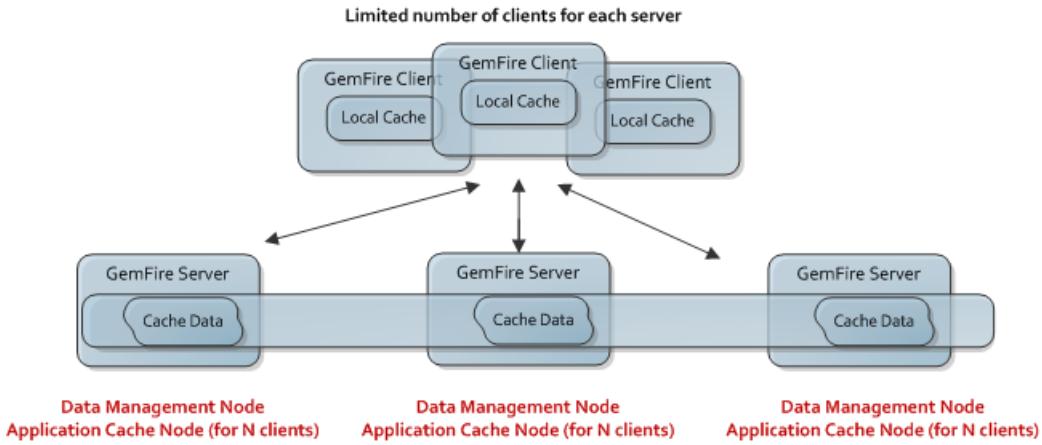
To install licensing for this topology, perform the following steps.

1. Obtain a Data Management Node license and Global WAN Upgrade license . You will receive two serial numbers from VMware. In a Global WAN deployment scenario, you can use the same two serial numbers in each distributed system; however, we recommend that you obtain a unique set of serial numbers for each distributed system communicating over the WAN.
2. Install the license in GemFire by using one of the following options:
  - Add the Data Management Node and Global WAN Upgrade serial numbers directly in gemfire.properties on each peer member in the distributed system. See [Option 1: Install Serial Numbers in gemfire.properties on page 55](#); or
  - Create a file that contains both the Data Management Node serial number and the Global WAN Upgrade serial number, and copy the file to the vFabric serial number directory on all peer or server members in the distributed system. See [Option 2: Install Serial Numbers in vFabric Serial Number Files on page 57](#); or
  - Install the Data Management Node license and Global WAN Upgrade using vFabric License Server. See [vFabric Suite Only: Configure GemFire for vFabric License Server on page 58](#).
3. If you have obtained unique serial numbers for your other distributed systems, install the unique serial numbers on the other distributed systems by using one of the options described in step 2.

### **Client/Server with a Limited Number of Clients**

A client/server system that only allows a specific number of clients requires an Application Cache Node serial number for the clients and a Data Management Node serial number for the servers. The serial numbers are installed on the servers only; you do not need to install any serial numbers on the GemFire clients.

The license quantity is embedded in the Application Cache Node serial number for the distributed system as a whole. License quantity reflects the number of 6-core CPUs that will be running clients. Each 6-core CPU can run multiple clients but will consume only one license.

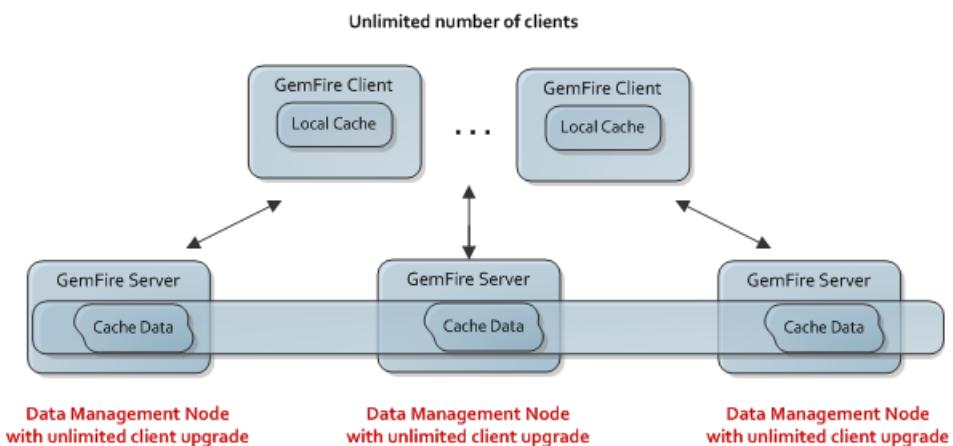


To install licensing for this topology, perform the following steps:

1. Obtain both a Data Management Node and Application Cache Node license from VMware for the desired quantity of servers and clients you wish to license.
2. After you have purchased the licenses, you will receive a Data Management Node serial number and an Application Cache Node serial number from VMware. Use one of the following options to install licensing in GemFire:
  - Add the Data Management Node license serial number and the Application Cache Node license serial number directly in `gemfire.properties` on each cache server member. See "Application Cache Node + Data Management Node Licenses" under [Option 1: Install Serial Numbers in gemfire.properties](#) on page 55; or
  - Create a file that contains the Data Management Node license serial number and a separate file that contains the Application Cache Node license serial number. Copy these files to the vFabric serial number directory on all cache server members. See [Option 2: Install Serial Numbers in vFabric Serial Number Files](#) on page 57; or
  - Install the Data Management Node and Application Cache Node licenses using vFabric License Server. See [vFabric Suite Only: Configure GemFire for vFabric License Server](#) on page 58.

### Client/Server with an Unlimited Number of Clients

A client/server system that allows an unlimited number of clients to connect to it requires a Data Management Node serial number and an Unlimited Client Upgrade serial number.



To install licensing for this topology, perform the following steps:

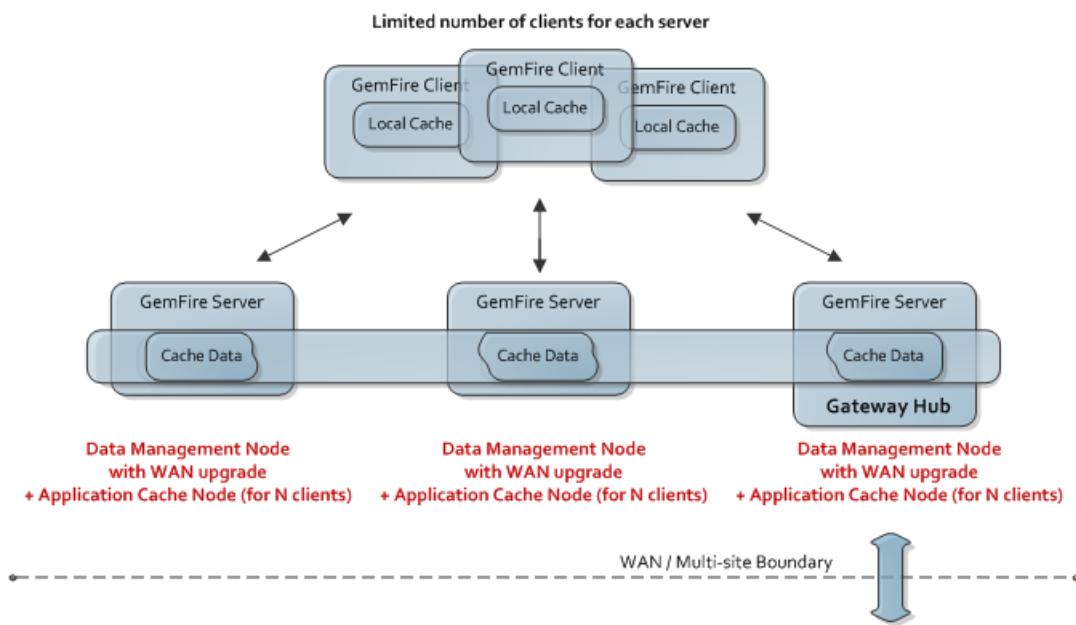
1. Obtain a Data Management Node license with Unlimited Client Upgrade.
2. After you purchase the license, you will receive two separate serial numbers. You must install both serial numbers in GemFire.
3. Use one of the following options to install the licensing in GemFire:
  - Add the Data Management Node license and Unlimited Client Upgrade serial numbers directly in gemfire.properties on each cache server member. See "Data Management Node with Upgrades Examples" under [Option 1: Install Serial Numbers in gemfire.properties](#) on page 55; or
  - Create a file that contains the Data Management Node license and Unlimited Client Upgrade serial numbers, and copy the file to the vFabric serial number directory on all cache server members. See "Data Management Node Licenses" under [Option 2: Install Serial Numbers in vFabric Serial Number Files](#) on page 57; or
  - Install the Data Management Node license and Unlimited Client Upgrade using vFabric License Server. See [vFabric Suite Only: Configure GemFire for vFabric License Server](#) on page 58.

### **Client/Server with Global WAN and a Limited Number of Clients**

A client/server deployment that only allows a specific number of clients and Global WAN enabled requires an Application Cache Node serial number (to specify the number of clients), a Data Management Node serial number, and a Global WAN upgrade serial number.



**Note:** You can use the same set of serial numbers on all distributed systems in a WAN deployment scenario; however, VMware recommends that you obtain a set of unique serial numbers for each distributed system that connects through the WAN. Each member of each distributed system communicating through the WAN must be installed with a license.



To install licensing for this topology, perform the following steps.

1. Obtain a Data Management Node and an Application Cache Node license from VMware for the desired quantity of servers and clients you wish to license. In addition, obtain a Global WAN Upgrade license. You will receive three serial numbers from VMware. In a Global WAN deployment scenario, you can use the

same three serial numbers in each distributed system; however, we recommend that you obtain a unique set of serial numbers for each distributed system communicating over the WAN.

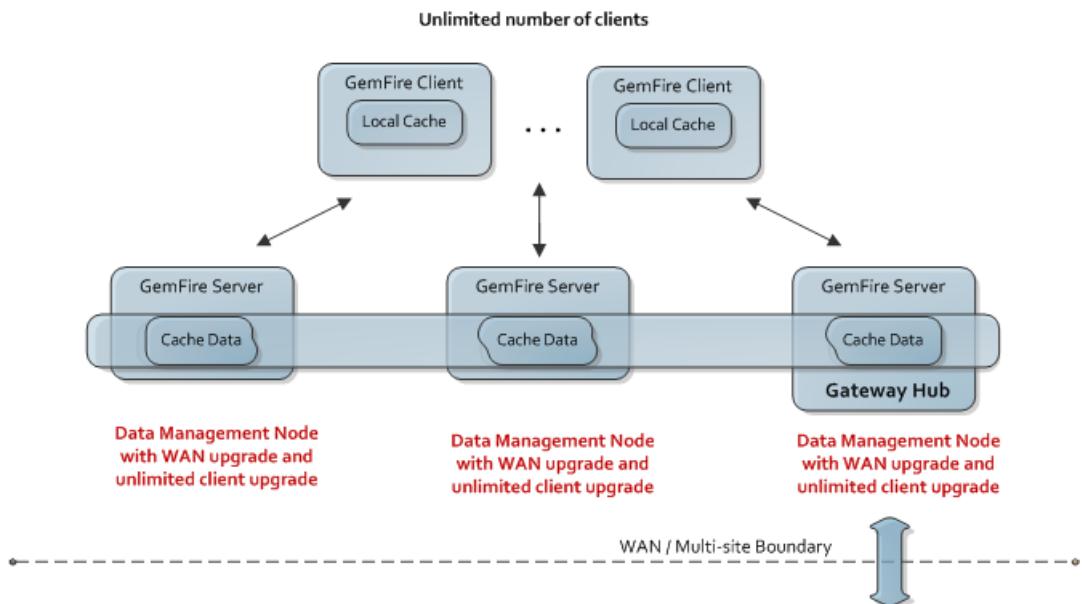
2. After you have purchased the licenses, you will receive a Data Management Node serial number, a Global WAN Upgrade serial number and an Application Cache Node serial number from VMware. Use one of the following options to install licensing in your distributed system:
  - Add the Data Management Node license serial number, the Global WAN Upgrade serial number and the Application Cache Node license serial number directly in `gemfire.properties` on each cache server member. See "Data Management Node with Application Cache Node Examples" under [Option 1: Install Serial Numbers in gemfire.properties](#) on page 55; or
  - Create a file that contains the Data Management Node license serial number and Global WAN Upgrade serial number and a separate file that contains the Application Cache Node license serial number. Copy these files to the vFabric serial number directory on all cache server members. See [Option 2: Install Serial Numbers in vFabric Serial Number Files](#) on page 57; or
  - Install the Data Management Node, Global WAN Upgrade and Application Cache Node licenses using vFabric License Server. See [vFabric Suite Only: Configure GemFire for vFabric License Server](#) on page 58.
3. If you have obtained unique serial numbers for your other distributed systems, install the unique serial numbers on the other distributed systems by using one of the options described in step 2.

### **Client/Server with Global WAN and Unlimited Clients**

A very common licensing option is the client/server with unlimited clients combined with WAN licensing topology. Client/Server with global WAN and unlimited clients requires a Data Management Node serial number, Global WAN upgrade serial number, and an Unlimited Clients serial number.



**Note:** You can use the same set of serial numbers on all distributed systems in a WAN deployment scenario; however, VMware recommends that you obtain a set of unique serial numbers for each distributed system that connects through the WAN. Each member of each distributed system communicating through the WAN must be installed with a license.



To install licensing for this topology, perform the following steps:

1. Obtain a Data Management Node license, a Global WAN Upgrade license and an Unlimited Clients Upgrade license. You will receive three serial numbers from VMware. In a Global WAN deployment scenario, you can use the same three serial numbers in each distributed system; however, we recommend that you obtain a unique set of serial numbers for each distributed system communicating over the WAN.
2. After you have received the serial numbers, use one of the following options to install the licensing in GemFire:
  - Add the three serial numbers directly in `gemfire.properties` on each cache server in the distributed system. See "Data Management Node with Upgrades Examples" under [Option 1: Install Serial Numbers in gemfire.properties](#) on page 55; or
  - Create a file that contains the three serial numbers, and copy the file to the vFabric serial number directory on all cache servers in the distributed system. See "Data Management Node Licenses" under [Option 2: Install Serial Numbers in vFabric Serial Number Files](#) on page 57; or
  - Install the Data Management Node license, the Global WAN Upgrade license and an Unlimited Clients license using vFabric License Server. See [vFabric Suite Only: Configure GemFire for vFabric License Server](#) on page 58
3. If you have received unique serial numbers for your other distributed systems, install the unique serial numbers on the other distributed systems by using one of the options described in step 2.

## Checking License Quantity

You can check the quantity of licenses in your vFabric GemFire serial number by using the License Check Tool.

To download the tool, visit the [vFabric GemFire product download page](#) and select the "Drivers and Tools" tab.

For instructions on how to use the tool, see the section "Check Existing License Key with the vFabric License Check Tool" at:

<http://pubs.vmware.com/vfabric51/topic/com.vmware.vfabric.platform.5.1/vfabric/install-activate-licenses.html>

## How GemFire Manages Licenses

Before you install vFabric GemFire licenses, understand how GemFire manages your license information.

This topic covers the following subjects:

- [How GemFire Finds and Verifies Your License](#) on page 52
- [License Working Directory](#) on page 53
- [Local VMware vFabric Directories](#) on page 53

### How GemFire Finds and Verifies Your License

GemFire has a default license that it uses if it cannot locate any other valid licensing.

Non-default licensing is verified using a combination of the product `gemfire.jar` file and the serial numbers you provide, or a license may be acquired dynamically from the vFabric License Server.

GemFire uses the first valid licensing it finds in this list:

1. Serial number specified by one of the license properties in `gemfire.properties`.
2. Dynamic licensing provided from a serial number file stored in the local VMware vFabric serial number directory, configured by specifying `dynamic` for one of the license properties in `gemfire.properties`.
3. Dynamic licensing provided by the vFabric License Server, configured by specifying `dynamic` for one or both of the license properties in `gemfire.properties`.

If GemFire cannot validate any of the specified licensing in the above list, the application or cache server process will not start and will throw an exception.

If no licensing is specified, GemFire uses the default evaluation licensing shipped with the product.

## **License Working Directory**

GemFire stores licensing information in a directory on your system.

GemFire writes to the first writable directory it finds in this list:

1. The value of the property `license-working-dir` if specified in the member's `gemfire.properties`.
2. The member process's current working directory as determined by `System.getProperty("user.dir")` at startup.

These are the files GemFire writes:

1. License state files with names ending with `-license.cfg`. Example: `vf.gf.dmn-license.cfg`.
2. License events files with names ending with `-events.txt`. Example: `vf.gf.dmn-events.txt`.

Leave these files alone. Do not edit or delete these files and do not alter the permissions on the files or the directory where these files are located. These files are created using the default permissions of the user who is starting up the GemFire process. Any user needing to stop or start the GemFire process will need write permissions for this directory and the files in it. Make sure that the permissions for each user are sufficient otherwise GemFire may throw an exception during stop or start.

## **Local VMware vFabric Directories**



**Note:** Directory path names are case sensitive. Please use the directory path names as specified in the tables below.

### **Local VMware vFabric home directory**

For most vFabric Suite components, the local VMware vFabric home directory is used as a license working directory by the vFabric License Manager. By default, however, vFabric GemFire uses the current working directory as the license working directory as described in [License Working Directory](#) on page 53. The default location of the local VMware vFabric home directory, if it exists, varies by operating system:

Windows	<code>%ALLUSERSPROFILE%\VMware\vFabric</code>
Linux (or other OS)	<code>/opt/vmware/vFabric</code>

### **Local VMware vFabric serial numbers directory**

The location of the local VMware vFabric serial numbers directory, if it exists, varies by operating system:

Windows	<code>%ALLUSERSPROFILE%\VMware\vFabric</code>
Linux (or other OS)	<code>/etc/opt/vmware/vfabric</code>

If the serial numbers directory does not already exist, you must first create the directory in order to use serial number license files for vFabric GemFire licensing. The directory should maintain read/write permissions for all users who will be starting and stopping GemFire processes.



# Chapter 11

---

## Installing and Configuring vFabric GemFire Licenses

You install or configure licensing for all members you run as peers or servers using the appropriate licensing for your deployment.

### Pre-Installation Notes

- You do not need to install licensing in your client applications.
- To understand your licensing options, including upgrade options, see [Choosing a License Option Based on Topology](#) on page 46.

The following sections describe your options for installing licenses.

### Option 1: Install Serial Numbers in gemfire.properties

Add the serial number(s) to the gemfire.properties file on each peer or cache server member, depending on your type of deployment. When adding multiple serial numbers, use a comma-delimited list (spaces between serial numbers are allowed).

This option is the recommended way to install licensing, except for GemFire deployments installed on vSphere virtual machines as part of a vFabric Suite deployment. In that case, see [vFabric Suite Only: Configure GemFire for vFabric License Server](#) on page 58. The following sections describe how to configure the properties file depending on your deployment topology:

- [Peer-to-Peer](#) on page 55
- [Peer-to-Peer with Global WAN](#) on page 56
- [Client/Server with a Limited or an Unlimited Number of Clients](#) on page 56
- [Client/Server with Global WAN and a Limited Number of Clients](#) on page 56
- [Client/Server with Global WAN and Unlimited Number of Clients](#) on page 57

#### Peer-to-Peer

**Example 1: Simple peer-to-peer.** Add an Application Cache Node serial number to the gemfire.properties file of each peer member.

```
# gemfire.properties for application cache peer  
license-application-cache=#####-#####-#####-#####-#####  
license-data-management=
```

**Example 2: Server peer-to-server peer.** Add a Data Management Node serial number to the `gemfire.properties` file of each cache server member.

```
# gemfire.properties for data management cache server
license-application-cache=
license-data-management=#####-#####-#####-#####-#####-#####


```

### Peer-to-Peer with Global WAN

Install the Data Management Node serial number and the Global WAN upgrade serial number in the `gemfire.properties` file of each peer member.



**Note:** In a Global WAN deployment scenario, you can use the same two serial numbers in each distributed system; however, VMware recommends that you obtain a unique set of serial numbers for each distributed system communicating over the WAN.

```
# gemfire.properties for data management cache server
# with global WAN
license-application-cache=
license-data-management=#####-#####-#####-#####-#####-#####
#####-#####-#####-#####-#####-#####-#####


```

### Client/Server with a Limited or an Unlimited Number of Clients

Add the Data Management Node serial number and additional serial number(s) to the `gemfire.properties` file of each cache server member.

**Example 1: Client/server with a limited number of clients.** Add a Data Management Node serial number and an Application Cache Node serial number (which specifies number of clients).

```
# gemfire.properties for data management cache server
# with a predetermined number of clients
license-application-cache=#####-#####-#####-#####-#####
license-data-management=#####-#####-#####-#####-#####
#####-#####-#####-#####-#####-#####


```

**Example 2: Client/server with an unlimited number of clients.** Add a Data Management Node serial number and an Unlimited Clients upgrade serial number.

```
# gemfire.properties for data management cache server
# with unlimited clients
license-application-cache=
license-data-management=#####-#####-#####-#####-#####
#####-#####-#####-#####-#####-#####


```

### Client/Server with Global WAN and a Limited Number of Clients

Add a Data Management Node serial number, a Global WAN Upgrade serial number, and an Application Cache Node serial number (which specifies the number of clients) to the `gemfire.properties` file of each cache server member.



**Note:** In a Global WAN deployment scenario, you can use the same two serial numbers in each distributed system; however, VMware recommends that you obtain a unique set of serial numbers for each distributed system communicating over the WAN.

```
# gemfire.properties for data management cache server
# with Global WAN and a limited number of clients
license-application-cache=#####-#####-#####-#####-#####
license-data-management=#####-#####-#####-#####-#####
#####-#####-#####-#####-#####


```

## Client/Server with Global WAN and Unlimited Number of Clients

Add a Data Management Node serial number, a Global WAN Upgrade serial number, and an Unlimited Clients upgrade serial number to the `gemfire.properties` file of each cache server member.



**Note:** In a Global WAN deployment scenario, you can use the same two serial numbers in each distributed system; however, VMware recommends that you obtain a unique set of serial numbers for each distributed system communicating over the WAN.

```
# gemfire.properties for data management cache server
# with unlimited clients and global WAN
license-application-cache=
license-data-management=#####-#####-#####-#####-#####
#####-#####-#####-#####-#####-#####
#####-#####-#####-#####-#####-#####
```

## Option 2: Install Serial Numbers in vFabric Serial Number Files

Another way for installing licenses is to add license files to the VMware vFabric serial number directory.

You can configure licensing by adding serial numbers to files in the VMware vFabric serial number directory and adding the keyword `dynamic` to the `license-application-cache` (Application Cache Node) and/or `license-data-management` (Data Management Node) property in `gemfire.properties`.

The following sections describe how to configure the serial number files and `gemfire.properties` file based on your deployment topology.

### Simple Peer-to-Peer

1. Create a file named `vf.gf.acn-serial-numbers.txt` and paste the Application Cache Node serial number into it.
2. Save this file to the appropriate serial numbers directory on each peer member. See [Local VMware vFabric Directories](#) on page 53 for the appropriate directory based on your operating system.
3. On all peer members, specify `license-application-cache=dynamic` in `gemfire.properties`. For example:

```
# gemfire.properties for dynamic licensing of
# application cache peer
license-application-cache=dynamic
license-data-management=
```

### Server Peer-to-Peer, Peer-to-Peer Over WAN, and Client/Server Deployments with Unlimited Clients

1. Create a file named `vf.gf.dmn-serial-numbers.txt` and paste the Data Management Node serial number(s) into it.



**Note:** If you have received multiple Data Management Node serial numbers for different upgrade options (Unlimited Clients upgrade or Global WAN upgrade), list each serial number on a new line. The list must contain one Data Management Node serial number with one or more Data Management Node upgrade serial numbers.

2. Save this file to the appropriate serial numbers directory on all cache servers or peer members. See [Local VMware vFabric Directories](#) on page 53 for the appropriate directory based on your operating system.
3. On all cache servers or peer members, specify `license-data-management=dynamic` in `gemfire.properties`. For example:

```
# gemfire.properties for dynamic licensing of
# data management cache server
```

```
license-application-cache=
license-data-management=dynamic
```

### **Client/Server with Limited Clients (WAN and non-WAN Deployments)**

1. Create both `vf.gf.dmn-serial-numbers.txt` and `vf.gf.acn-serial-numbers.txt` as described in the two previous procedures.
2. Save the files to the appropriate serial numbers directory on each cache server member.
3. Revise your `gemfire.properties` file on all cache servers:

```
# gemfire.properties for dynamic licensing of
# data management cache server and application cache peers
license-application-cache=dynamic
license-data-management=dynamic
```

## **vFabric Suite Only: Configure GemFire for vFabric License Server**

If you are running vFabric GemFire on a vSphere virtual machine as part of a vFabric Suite package, configure vFabric GemFire to communicate with the vFabric License Server.

If you have not done so, install the vFabric License Server and activate your licenses as described in *Getting Started with vFabric Suite*. Then add the keyword "dynamic" to the appropriate license property field in the `gemfire.properties` file on all nodes.

The following sections describe how to configure the properties files based on your deployment topology:

- [Simple Peer-to-Peer](#) on page 58
- [Server Peer-to-Peer, Peer-to-Peer Over WAN, and Client/Server with Unlimited Clients](#) on page 58
- [Client/Server with Limited Clients \(WAN and non-WAN\)](#) on page 59
- [Optional Configuration Change: Change Default Timeout Value \(10000\)](#) on page 59

### **Simple Peer-to-Peer**

**Application Cache Node license.** Specify `license-application-cache=dynamic` in `gemfire.properties` on each cache peer member of the system.

For example:

```
# gemfire.properties for dynamic licensing of
# application cache peer
license-application-cache=dynamic
license-data-management=
```

### **Server Peer-to-Peer, Peer-to-Peer Over WAN, and Client/Server with Unlimited Clients**

**Data Management Node license (with or without upgrades).** Specify `license-data-management=dynamic` in `gemfire.properties` on each cache server member of the system.

For example:

```
# gemfire.properties for dynamic licensing of
# data management cache server
license-application-cache=
license-data-management=dynamic
```

## Client/Server with Limited Clients (WAN and non-WAN)

**Application Cache Node and Data Management Node licenses.** For deployments with a predetermined number of clients and where you want to use dynamic licensing, you specify `dynamic` for both `license-application-cache` and `license-data-management` in `gemfire.properties`.

For example:

```
# gemfire.properties for dynamic licensing of
# data management cache server
license-application-cache=dynamic
license-data-management=dynamic
```

## Optional Configuration Change: Change Default Timeout Value (10000)

In `gemfire.properties`, change the default timeout value (10000) to indicate the maximum time in milliseconds that the members should wait when obtaining a license dynamically from the vFabric License Server. For example :

```
#timeout in milliseconds
license-server-timeout=20000
```

## Troubleshooting Issues with vFabric GemFire Licenses

This section provides some basic troubleshooting on vFabric license installation.

### Basic Troubleshooting Steps

If you install an invalid serial number or if GemFire is unable to obtain a dynamic license from the vFabric License Server, GemFire will fail to start and will throw an exception. In this case, depending on your licensing configuration, you may need to perform one of the following fixes:

1. Replace the invalid serial number or serial numbers specified either in `gemfire.properties` or in serial number files with valid (non-expired) serial licenses, and restart the server.
2. Completely remove the invalid serial number or serial numbers from `gemfire.properties`, and restart the server. When the server restarts, it will use the default evaluation license.
3. Remove the keyword "dynamic" from `gemfire.properties`, and restart the server. When the server restarts, it will use the default evaluation license
4. Increase the amount of time specified in `license-server-timeout`. This option is only applicable if you are running GemFire on a vSphere virtual machine and using the vFabric License Server to acquire licenses dynamically.

For more details on licensing installation issues, see [Application or cache server process does not start](#) on page 607 in the *Troubleshooting and System Recovery* section.



# Chapter 12

---

## Setting Up the Product Examples

The GemFire examples are included in the zip file that contains your GemFire software. Set up your environment to run these examples as described in this section.

GemFire includes an evaluation license that enables you to run the examples and perform simple evaluation activities. This license allows you to run up to three peer or cache server members and up to three clients for each cache server member. GemFire uses the default evaluation license when no other licensing is installed. The default evaluation license never expires. It is not allowed for production use.

### Prerequisites

- Install GemFire on each physical or VMware virtual machine on which you will run the vFabric GemFire examples as described in [Installing vFabric GemFire](#) on page 27.
- Make sure you have set your JAVA\_HOME, GEMFIRE, and GF\_JAVA environment variables properly configured.

### Procedure

1. Locate the SampleCode directory in your VMware vFabric GemFire product installation. (See [Table 2: GemFire Examples](#) on page 62.)
2. Add the example class directories to your CLASSPATH.

- tutorial/classes
- helloworld/classes
- quickstart/classes
- examples/dist/classes

In the example class directories, replace <InstallDirectory> with the directory that contains your GemFire product installation. For example:

- **Unix Bourne and Korn shells (sh, ksh, bash).**

```
CLASSPATH=$CLASSPATH:\$GEMFIRE/SampleCode/tutorial/classes:\$GEMFIRE/SampleCode/helloworld/classes:\$GEMFIRE/SampleCode/quickstart/classes:\$GEMFIRE/SampleCode/examples/dist/classes:\$CLASSPATH;export CLASSPATH
```

- **Windows**

```
set CLASSPATH=%CLASSPATH%;%GEMFIRE%\tutorial\classes;%CLASSPATH%\SampleCode\helloworld\classes;%CLASSPATH%\SampleCode\quickstart\classes;%CLASSPATH%\SampleCode\examples\dist\classes
```

## Examples Included with vFabric GemFire

The following examples are included in the `SampleCode` directory from your GemFire product download.

**Table 2: GemFire Examples**

Example	Example Code Location	Documentation Location
<b>Hello World</b> A simple demonstration of GemFire data distribution.	<code>SampleCode/helloworld</code>	vFabric GemFire User's Guide: <a href="#">Hello World Example</a> on page 81
<b>vFabric GemFire Tutorial</b> A guided tour of vFabric GemFire's core features.	<code>SampleCode/tutorial</code>	vFabric GemFire User's Guide: <a href="#">vFabric GemFire Tutorial</a> on page 65
<b>QuickStart Examples</b> Demonstrates the following GemFire features: <ul style="list-style-type: none"><li>• Benchmark examples:<ul style="list-style-type: none"><li>• Peer to peer</li><li>• Client-server</li></ul></li><li>• Cache eviction</li><li>• Cache expiration</li><li>• Delta propagation</li><li>• Distributed locking</li><li>• Fixed partitioning</li><li>• Function execution</li><li>• JSON</li><li>• Management and monitoring</li><li>• Multiple secure client</li><li>• Overflowing data to disk</li><li>• Partitioned data caching</li><li>• Persisting data</li><li>• Querying</li><li>• Replicated caching</li></ul> <li>• Reliable event notifications:<ul style="list-style-type: none"><li>• Interest registration</li><li>• Continuous querying</li><li>• Durable event messaging</li></ul></li> <li>• Secure client</li> <li>• Server-managed caching</li> <li>• Transactions</li>	<code>SampleCode/quickstart</code>	vFabric GemFire User's Guide: <a href="#">QuickStart Examples</a> on page 85
<b>Programming Examples</b> The following examples demonstrate how to write applications using the vFabric GemFire Java API: <ul style="list-style-type: none"><li>• Cache overflow</li></ul>	<code>SampleCode/examples/dist</code>	Documentation for the Programming Examples is available in the following files in the <code>SampleCode</code> directory of your GemFire download:  Examples guide:

Example	Example Code Location	Documentation Location
<ul style="list-style-type: none"> <li>• Cache runner</li> <li>• Client API</li> <li>• Client/Server configurations</li> <li>• Delta propagation</li> <li>• Flash cache</li> <li>• High availability overflow</li> <li>• Partitioned persistence</li> <li>• Portable data exchange serialization (PDX)</li> <li>• System administration explorer</li> <li>• Transactions</li> <li>• WAN</li> <li>• WAN: active data</li> </ul>		<p>SampleCode/examples/ README_dist_index.html (Each example subdirectory contains a README.html file with the documentation for that example.)</p> <p>Examples Javadocs: SampleCode/examples/dist/ javadoc/index.html</p>
<b>OSGi Examples</b> The following examples demonstrate how to deploy GemFire applications into an OSGi environment: <ul style="list-style-type: none"> <li>• cacheWorker</li> <li>• cacheWorkerFragment</li> </ul>	SampleCode/examples/osgi	SampleCode/examples/ osgi/dist/README.html

## What to do next

- Run through the vFabric GemFire Tutorial. See [vFabric GemFire Tutorial](#) on page 65.
- Explore GemFire configuration files and application code. See [Hello World Example](#) on page 81.
- Run the examples. See [QuickStart Examples](#) on page 85.



**Note:** The [Spring Data GemFire](#) project enables developers to use the Spring Data GemFire API to write applications and configure a GemFire system. See [Spring Data GemFire Code Examples](#) on page 119.



# Chapter 13

---

## vFabric GemFire Tutorial

The tutorial is a guided tour of vFabric GemFire's core feature set.

### Tutorial Introduction

The tutorial application demonstrates basic features of vFabric GemFire by walking through the code of a rudimentary social networking application built on GemFire.

The tutorial shows you how to kill JVMs without loss of service, add more storage dynamically while the application is running, and provide low latency access to your data.



**Note:** The tutorial demonstrates how to use these features with GemFire's Java API. If you plan to use C++ or C#, you may prefer to get started with the native client documentation. See <https://www.vmware.com/support/pubs/vfabric-gemfire.html> for a link to the most recent native client documentation.

### vFabric GemFire Features Demonstrated in the Tutorial

Use the tutorial to explore vFabric GemFire's main features.

GemFire distributed system	JVMs running GemFire form a distributed system. Each JVM is a GemFire peer member in the distributed system. A peer member connects to a GemFire distributed system and shares data with other peer members through data regions that are configured to distribute events among themselves.  To start a GemFire peer, you create a GemFire cache in each member. The cache manages the connectivity to other GemFire peers. Peers discover each other through multicast messaging or a TCP locator service.
Regions	Regions are an abstraction on top of the distributed system. A region lets you store data in many JVMs in the system without regard to which peer's memory the data is stored in. Regions give you a map interface that transparently fetches your data from the appropriate cache. The Region class extends the java.util.Map interface, but it also supports querying and transactions.
Replicated regions	A replicated region keeps a full copy of the region on each GemFire peer.
Partitioned regions	Partitioned regions allow you to configure the number of copies of data in your distributed system. GemFire partitions your data so that each peer only stores a part of the region contents.
Client caching	A GemFire distributed system is a mesh network, where all peers are connected directly to all other peers. GemFire also supports clients of the distributed system, which are connected only to a few of the peers in the distributed system. When peers are configured for client connections, they become servers to the clients as well as peers. Clients can have their own local cache of data from their server's distributed system, and they can update their local cache by registering with the server to receive changes to the data. GemFire also provides C++ and C# client APIs.

Shared-nothing  
persistence

GemFire supports shared-nothing persistence, where each peer persists its partition of the data set to its local disk. GemFire persistence also allows you to maintain a configurable number of copies of your data on disk.

## Sample Social Networking Application

The GemFire tutorial uses a sample social networking application. To support this application, the tutorial includes two sample regions and two sample application classes.

To store the sample social networking data the application uses two regions; one holds people and the other holds posts.

**Table 3: People Region**

Entry field	key	value
Type	String	com.gemstone.gemfire.tutorial.model.Profile
Description	The person's name	The profile of this person

**Table 4: Posts region**

Entry field	key	value
Type	com.gemstone.gemfire.tutorial.model.PostID	String
Description	A unique ID for this post	The text of the post

The Profile class holds a set of friends belonging to one person. A PostID is an author's name and a timestamp. The author's name should match a key (the person's name) in the people region, but that constraint is not enforced in this simple application.

Profile	PostID
<pre>+friends: Set&lt;String&gt; +addFriend(name:String): void +removeFriend(name:String): void +getFriends(): Set&lt;String&gt;</pre>	<pre>+author: String +timestamp: long +getAuthor(): String +getTimestamp(): long</pre>

A simple command line UI enables you to add people and posts to the system.

## Running the Tutorial

The tutorial shows a sample social networking application that stores people's profiles and posts. Each person has a profile that lists their friends, and each person can write posts.

Running the tutorial encompasses the following areas:

- [Prerequisites](#) on page 67
- [Part 1: Start a Locator](#) on page 67
- [Part 2: Create a Cache and Start Peers](#) on page 67
- [Part 3: Create Replicated Regions](#) on page 68
- [Part 4: Create Partitioned Regions](#) on page 70
- [Part 5: Set Up Client/Server Caching](#) on page 71
- [Part 6: Add and Stop Cache Servers \(Peers\)](#) on page 75
- [Part 7: Configure Persistence](#) on page 76

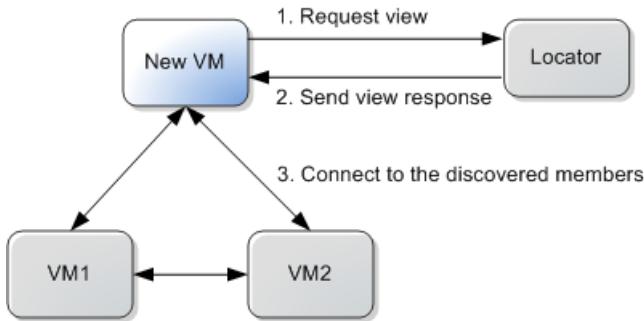
## Prerequisites

Before you begin, verify that you meet the prerequisites:

- Install and configure vFabric GemFire and the product examples. See [Installing vFabric GemFire](#) on page 27 and [Setting Up the Product Examples](#) on page 61.
- Verify that your system has JDK 1.6 or later installed. The gfsh command-line utility also requires that the tools.jar included with the JDK is added to your CLASSPATH. The gfsh scripts add \$JAVA\_HOME/lib/tools.jar (or %JAVA\_HOME%\lib\tools.jar for Windows) to your CLASSPATH.
- Make sure that your CLASSPATH includes \$GEMFIRE/SampleCode/tutorial/classes or %GEMFIRE%\SampleCode\tutorial\classes.
- Open several terminal sessions.

### Part 1: Start a Locator

JVMs running GemFire discover each other through multicast messaging or through a TCP locator service, which is called a locator. Multicasting is a good way to run quick tests, but it is less robust and flexible than the locator service. The locator runs as a separate process to which each new member connects to first discover the list of available peers. Clients connect to the locator to get server connection information. For more information, see [How Member Discovery Works](#) on page 165. For this example, we use a locator.



**Figure 1: Peer Discovery Using a Locator**

#### 1. Start a locator.

First, change directories to the tutorial directory, <product\_directory>/SampleCode/tutorial. For example:

```
$ cd <product_directory>/SampleCode/tutorial
```

where <product\_directory> corresponds to the location where you installed vFabric GemFire.

Type the following command at the prompt:

```
$ gfsh start locator --name=locator1 --port=55221
```

The locator process runs in the background, listening for connections on port 55221. To stop the process, you can type gfsh stop locator --dir=locator1. But don't stop it yet.

### Part 2: Create a Cache and Start Peers

You will store the data on several GemFire peers. The first step to starting up a GemFire peer is to create a com.gemstone.gemfire.cache.Cache. The cache is the central component of GemFire. It manages connections to other GemFire peers. For more information, see [Cache Management](#) on page 137.

1. Use a text editor or your favorite IDE to open the GemfireDAO.java file in the tutorial application code. This file is located <product\_directory>/SampleCode/tutorial/src/com/gemstone/gemfire/tutorial/storage directory where <product\_directory> corresponds to the location where you installed vFabric GemFire.

Look at the cache creation in the initPeer method in GemfireDAO.java. The first thing this method does is create a cache:

```
Cache cache = new CacheFactory()
    .set("locators", "localhost[55221]")
    .set("mcast-port", "0")
    .set("log-level", "error")
    .create();
```

Secondly, it configures the cache:

- The GemFire locators property tells the cache which locators to use to discover other GemFire JVMs.
- The mcast-port property tells GemFire not to use multicast discovery to find peers.
- The log-level property controls the log level of GemFire's internal logging. Here it is set to "error" to limit the amount of messaging that will show up in the console.

When this code is run, after the call to create finishes, this peer will discover other peers and connect to them.

2. **Run the Peer application in two terminal sessions.** You already have one window open where you started the locator. Start another terminal window. In each window, run the Peer application:

```
$ java com.gemstone.gemfire.tutorial.Peer
```

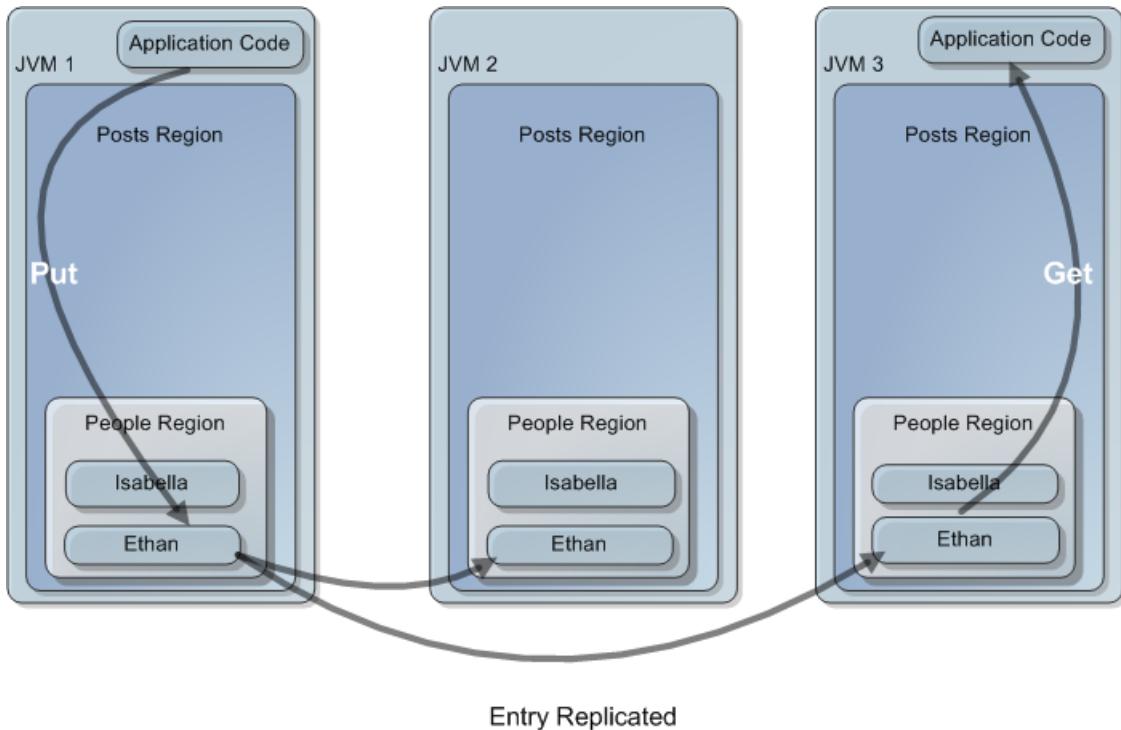
The peers start up and connect to each other. You should see the interactive command prompt for the tutorial's social networking application in each window:

```
Commands:
person [name] [friend] [friend] ...
    Add a person. The person's name and friends cannot
    contain spaces
post [author] [text]
    Add a post.
people
    List all people.
posts
    List all posts.
quit
    Exit the shell.
>
```

### Part 3: Create Replicated Regions

The GemFire com.gemstone.gemfire.cache.Region interface defines a key-value collection. Region extends the java.util.concurrent.ConcurrentMap interface. The simplest type of region is a replicated region. Every peer that hosts a replicated region stores a copy of the entire region locally. Changes to the replicated region are sent synchronously to all peers that host the region. For more information on regions, see [Data Regions](#) on page 145.

This procedure walks you through the creation of a replicated region called People.



Perform the following steps:

- Look at the initPeer method in GemfireDAO.java to see where it creates the people region.**

To create a `com.gemstone.gemfire.cache.Region`, you use a `com.gemstone.gemfire.cache.RegionFactory` class. This is the code in `initPeer` method that creates the people region:

```
people = cache.<String, Profile>createRegionFactory(REPLICATE)
    .addCacheListener(listener)
    .create("people");
```

The people region is constructed with `com.gemstone.gemfire.cache.RegionShortcut.REPLICATE`, which tells the factory to start with the configuration for a replicated region. This method adds a cache listener to the region. You can use a cache listener to receive notifications when the data in the region changes. This sample application includes a `LoggingCacheListener` class, which prints changes to the region to `System.out` and lets you see how entries are distributed.

- Look at the addPerson method.** It adds the entry to the region by calling the `put` method of `Region`.

```
public void addPerson(String name, Profile profile) {
    people.put(name, profile);
}
```

Calling `put` on the people region distributes the person to all other peers that host that region. After this call completes, each peer will have a copy of this person.

- Add people.** In one of your terminal windows, type:

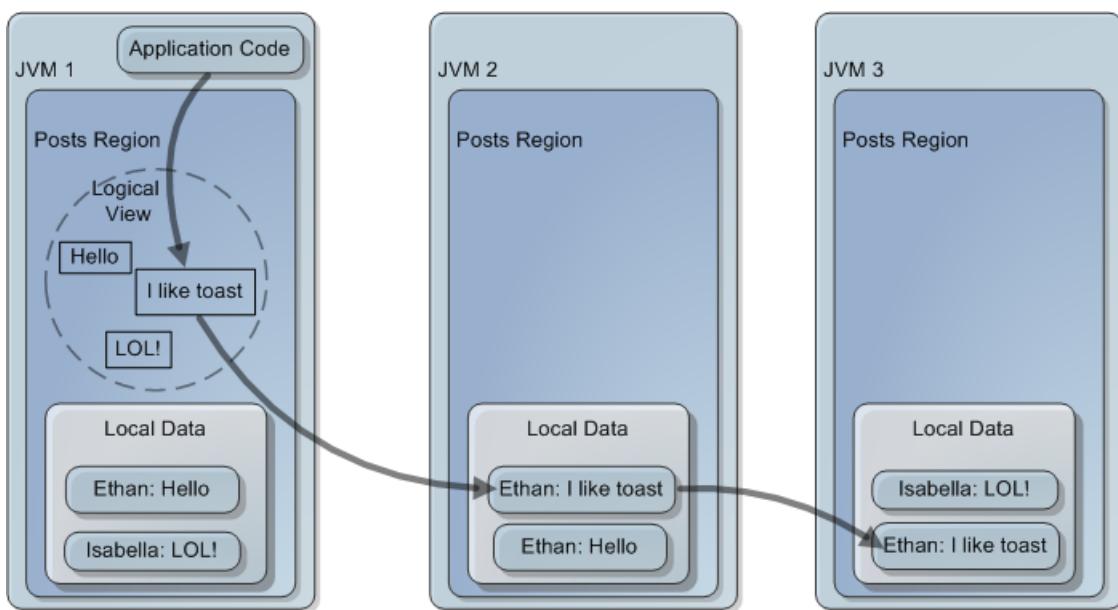
```
person Isabella
person Ethan
```

You will see the users show up in the other window:

```
In region people created key Isabella value Profile [friends=[]]
In region people created key Ethan value Profile [friends=[]]
```

## Part 4: Create Partitioned Regions

You expect to create a lot of posts. Because of that, you do not want to host a copy of the posts on every server. Thus you store them in a partitioned region. The API to use the partitioned region is the same as that for a replicated region, but the data is stored differently. A partitioned region lets you control how many copies of your data will exist in the distributed system. The data is partitioned over all peers that host the partitioned region. GemFire automatically maintains the number of copies of each partition that you request.



In the above diagram, the application code writes the post "I like toast" to two JVMs; it stores a primary copy in one peer (JVM 2) and a redundant copy in another peer (JVM 3). You will also notice that while each post appears in the local data of two peers, the logical view contains all three posts.

1. **Look at the code that creates the posts region.** You can create partitioned regions with the PARTITION\_XXX shortcuts.

To create the posts region, the `initPeer` method uses the `com.gemstone.gemfire.cache.RegionShortcut.PARTITION_REDUNDANT` shortcut to tell GemFire to create a partitioned region that keeps one primary and one redundant copy of each post on different machines.

```
posts = cache.<PostID, String>createRegionFactory(PARTITION_REDUNDANT)
    .addCacheListener(listener)
    .create("posts");
```

2. **Start another terminal window and launch the peer application in that window.**

```
$ java com.gemstone.gemfire.tutorial.Peer
```

You should have three peers running now. Each post you create will be stored in only two of these peers.

**3. Add some posts to the posts region.**

```
> post Isabella I like toast
> post Isabella LOL!
> post Ethan Hello
```

You see that the listener in only one of the JVMs is invoked for each post. For example, in only one of your open terminal windows, a message for each of your posts similar to the following appears:

```
> In region posts created key PostID [author=Isabella,
timestamp=1375987268621]
value I like toast
```

That's because partitioned regions make one copy of the post the primary copy. By default GemFire only invokes the listener in the peer that holds the primary copy of each post.

**4. From any window, list the available posts with the posts command.**

You should be able to list all posts, because GemFire fetches them from the peer that hosts each post.

```
> posts
Isabella: I like toast
Ethan: Hello
Isabella: LOL!
```

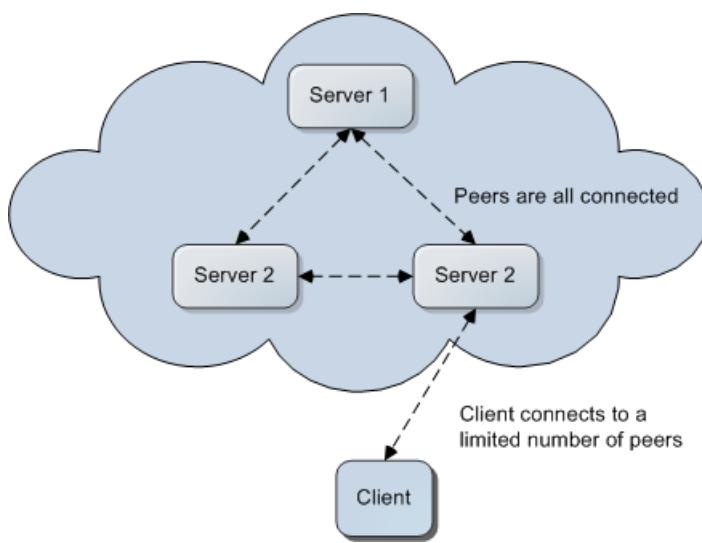
If you kill one of the JVMs, you should still be able to list all posts. You can bring that member back up and kill another one, and you should still see all of the posts.

**5. Kill your peers before moving on to the next section.** Type quit in each window.

## Part 5: Set Up Client/Server Caching

You have a fully working system now, but the UI code is running in the same member in which you are storing data. That works well for some use cases, but it may be undesirable for others. For example, if you have a Web server for the UI layer, you may want to increase or decrease the number of Web servers without modifying your data servers. Or you may need to host only 100 GB of data, but have thousands of clients that might access it. For these use cases, it makes more sense to have dedicated GemFire servers and access the data through a GemFire client.

GemFire servers are GemFire peers, but they also listen on a separate port for connections from GemFire clients. GemFire clients connect to a limited number of these cache servers.



Like regular peers, GemFire servers still need to define the regions they will host. You could take the peer code you already have and create a `com.gemstone.gemfire.cache.server.CacheServer` instance programmatically, but this example uses the `cacheserver` script that ships with GemFire. The `cacheserver` script reads the cache configuration from a cache xml file, which is a declarative way to define the regions in the cache.

- 1. Walk through configuring GemFire with XML.** All GemFire configuration that you can do in java you can also do in xml. Look at the `server.xml` file located in the `<product_directory>/SampleCode/tutorial/xml` directory where `<product_directory>` corresponds to the location where you installed vFabric GemFire. This file creates the same replicated and partitioned regions as the java code in the `GemfireDAO.initPeer` method, demonstrated in the preceding procedure.

```
?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>
  <region name="people" refid="REPLICATE">
    <region-attributes>
      <cache-listener>
        <class-name>
          com.gemstone.gemfire.tutorial.storage.LoggingCacheListener
        </class-name>
      </cache-listener>
    </region-attributes>
  </region>
  <region name="posts" refid="PARTITION_REDUNDANT">
    <region-attributes>
      <cache-listener>
        <class-name>
          com.gemstone.gemfire.tutorial.storage.LoggingCacheListener
        </class-name>
      </cache-listener>
    </region-attributes>
  </region>
</cache>
```



**Note:** The tutorial's sample XML files reference a cache DTD file from an earlier GemFire version. All GemFire schemas from previous GemFire versions and the current GemFire version are included with the product.

- 2. From the tutorial directory, start two cache servers.**

First, change directories to the tutorial directory, `<product_directory>/SampleCode/tutorial`. For example:

```
$ cd <product_directory>/SampleCode/tutorial
```

where `<product_directory>` corresponds to the location where you installed vFabric GemFire.

Type the following command at the prompt:

```
$ gfsh start server --name=server1 --locators=localhost[55221]
--cache-xml-file=../xml/server.xml --server-port=0
```

This command will automatically create a working directory for this server named `server1` under the `<product_directory>/SampleCode/tutorial` directory where you executed the command.

Next start a second cache server by executing the following command in the same terminal:

```
$ gfsh start server --name=server2 --locators=localhost[55221]
--cache-xml-file=../xml/server.xml --server-port=0
```

This command will automatically create a working directory for this server named `server2` under the `<product_directory>/SampleCode/tutorial` directory where you executed the command.

You should now have two GemFire peers (cache servers) that are listening for client connections. Here is what all those command line parameters mean.

**--name**

Name to give to the server instance. The command will also create a directory and log files named after this server instance.

**--locators**

List of locator hosts and ports used to discover other cache servers.

**--cache-xml-file**

Where to find the cache xml file to use, relative to the working directory of the server.

**--server-port**

Port on which to listen for GemFire clients. 0 means the server will listen on an ephemeral port, which is a temporary port assigned to the process by the OS.

3. **Use a text editor or your favorite IDE to open the GemfireDAO.java file in the tutorial application code.** This file is located `<product_directory>/SampleCode/tutorial/src/com/gemstone/gemfire/tutorial/storage` directory where `<product_directory>` corresponds to the location where you installed vFabric GemFire.

a. **Look at the code in GemfireDAO.java to see how it starts a client.**

Starting a GemFire client is very similar to starting a GemFire peer. In the GemFire client, you create an instance of `com.gemstone.gemfire.cache.client.ClientCache`, which connects to the locator to discover servers.

Examine the `GemfireDAO.initClient` method. The first thing the method does is create a `ClientCache`:

```
ClientCache cache = new ClientCacheFactory()
    .addPoolLocator("localhost", 55221)
    .setPoolSubscriptionEnabled(true)
    .setPoolSubscriptionRedundancy(1)
    .set("log-level", "error")
    .create();
```

Once you create a `ClientCache`, it maintains a pool of connections to the servers similar to a JDBC connection pool. However, with GemFire you do not need to retrieve connections from the pool and return them. That happens automatically as you perform operations on the regions. The pool locator property tells the client how to discover the servers. The client uses the same locator that the peers do to discover cache servers. Setting the subscription enabled and subscription redundancy properties allow the client to subscribe to updates for entries that change in the server regions. You are going to subscribe to notifications about any people that are added. The updates are sent asynchronously to the client. Because the updates are sent asynchronously, they need to be queued on the server side. The subscription redundancy setting controls how many copies of the queue are kept on the server side. Setting the redundancy level to 1 means that you can lose 1 server without missing any updates.

b. **Look at the code that creates a proxy region called posts in the client.**

Creating regions in the client is similar to creating regions in a peer. There are two main types of client regions, PROXY regions and CACHING\_PROXY regions. PROXY regions store no data on the client. CACHING\_PROXY regions allow the client to store keys locally on the client. This example uses a lot of posts, so you won't cache any posts on the client. You can create a proxy region with the shortcut PROXY. Look at the `GemfireDAO.initPeer` method. This method creates the posts region like this:

```
posts = cache.<PostID, String>createClientRegionFactory(Proxy)
    .create("posts");
```

c. **Look at the code that creates a caching proxy region called people in the client.**

You do not have many people, so for this sample the client caches all people on the client. First you create a region that has local storage enabled. You can create a region with local storage on the client with ClientRegionShortcut.CACHING\_PROXY. In the initClient method, here's where the people region is created.

```
people = cache.<String,
Profile>createClientRegionFactory(CACHING_PROXY)
    .addCacheListener(listener)
    .create("people");
```

d. **Look at the code that calls the registerInterest method to subscribe to notifications from the server.**

By creating a CACHING\_PROXY, you told GemFire to cache any people that you create from this client. However, you can also choose to receive any updates to the people region that occur on other peers or other clients by invoking the registerInterest methods on the client. In this case you want to register interest in all people, so you cache the entire people region on the client. The regular expression ".\*" matches all keys in the people region.

Look at the initClient method. The next line calls registerInterestRegex:

```
people.registerInterestRegex( ".*" );
```

When the registerInterestRegex method is invoked, the client downloads all existing people from the server. When a new person is added on the server it is pushed to the client.

e. **Look at the code that iterates over keys from the client.**

Look at the getPeople and getPosts methods in GemFireDAO.

```
public Set<String> getPeople() {
    return people.keySet();
}

public Set<PostID> getPosts() {
    if(isClient) {
        return posts.keySetOnServer();
    } else {
        return posts.keySet();
    }
}
```

GemFireDAO.getPeople calls people.keySet(), whereas GemFireDAO.getPosts calls posts.keySetOnServer() for the client. That's because the keySet method returns the keys that are stored locally on the client. For the People region you can use your locally cached copy of the keys, but for the Post region you need to go to the server to get the list of keys.

**4. Open two terminal windows, and start the client application in both terminals.**

Type the following command to start the client:

```
$ java com.gemstone.gemfire.tutorial.Client
```

Add people and posts from one of the clients. For example, add a person:

```
>person Ethan Isabella
```

Notice that the following message appears in both clients:

```
In region people created key Ethan value Profile [friends=[Isabella]]
```

In either one of the client windows, add more people and add some posts:

```
>person Isabella Ethan
>post Ethan Waaa!!!
>post Isabella I want toast
```

Switch to the other client window, and observe that the data you entered with one client can be seen by the other client:

```
>people
Ethan
Isabella
>posts
Ethan: Waaa!!!
Isabella: I want toast
```

Leave the clients running for the next part.

## Part 6: Add and Stop Cache Servers (Peers)

You can dynamically add peers to the system while it is running. The new peers are automatically discovered by the other peers and the client. The new peers automatically receive a copy of any replicated regions they create. However, partitioned regions do not automatically redistribute data to the new peer unless you explicitly instruct GemFire to rebalance the partitioned region. With the gfsh start server command, you can pass a command line flag indicating that the new peer should trigger a rebalance of all partitioned regions.

### 1. Add a new peer.

Open a terminal, and change directories to the tutorial directory,

`<product_directory>/SampleCode/tutorial`. For example:

```
$ cd <product_directory>/SampleCode/tutorial
```

where `<product_directory>` corresponds to the location where you installed vFabric GemFire.

Type the following command at the prompt:

```
$ gfsh start server --name=server3 --locators=localhost[55221]
--cache-xml-file=../xml/server.xml --server-port=0 --rebalance
```

This command will automatically create a working directory for this server named `server3` under the `<product_directory>/SampleCode/tutorial` directory where you executed the command.

### 2. Stop the cache servers.

#### a. Stop the first cache server::

```
$ gfsh stop server --dir=server1
```

b. Observe that the data you entered is still available and the client automatically ignores the dead server.

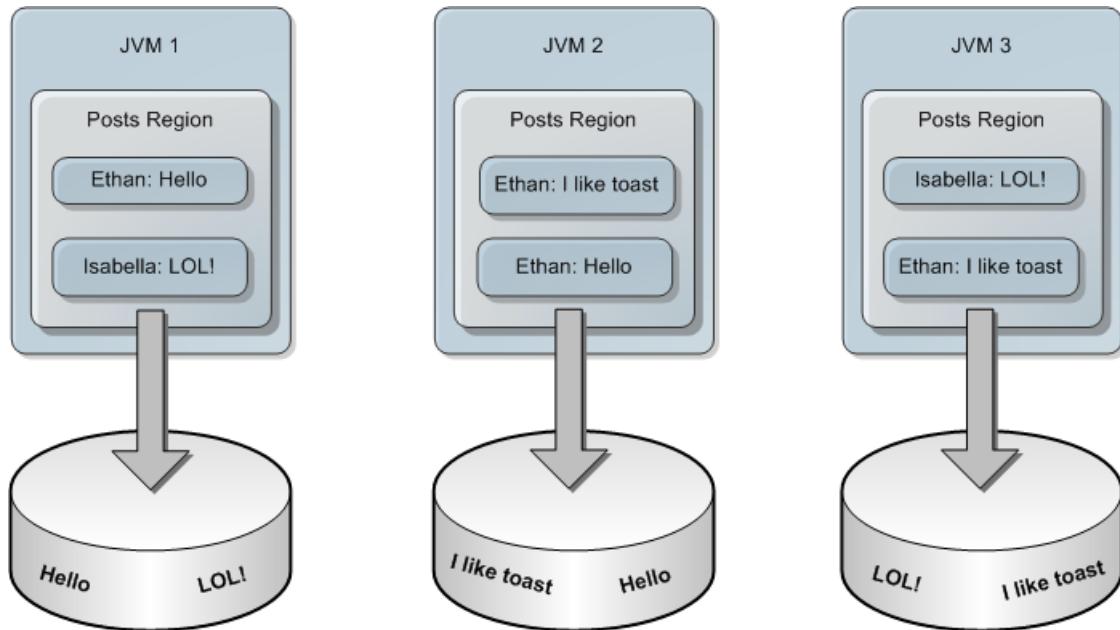
```
>people
Ethan
Isabella
>posts
Ethan: Waaa!!!
Isabella: I want toast
```

c. Stop the other cache servers. You can leave the client running.

```
$ gfsh stop server --dir=server2
$ gfsh stop server --dir=server3
```

## Part 7: Configure Persistence

GemFire supports shared-nothing persistence. Each member writes its portion of the region data to its own disk files. For the People region, each peer will be writing the entire region to its own disk files. For the Posts region, a copy of each post will reside on two different peers.



Each member persists its local data to its own disk.

### 1. Walk through configuring a persistence region.

Look at the `<SamplesDirectory>/xml/persistent_server.xml` file (where `<SamplesDirectory>` corresponds to the directory where you unzipped the example code files). It's exactly the same as the `server.xml` file, except that each region uses the `XXX_PERSISTENT` shortcut.

```
<?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>
  <region name="people" refid="REPLICATE_PERSISTENT">
    <region-attributes>
      <cache-listener>
        <class-name>
          com.gemstone.gemfire.tutorial.storage.LoggingCacheListener
        </class-name>
      </cache-listener>
    </region-attributes>
  </region>
  <region name="posts" refid="PARTITION_REDUNDANT_PERSISTENT">
    <region-attributes>
      <cache-listener>
        <class-name>
          com.gemstone.gemfire.tutorial.storage.LoggingCacheListener
        </class-name>
      </cache-listener>
    </region-attributes>
  </region>
</cache>
```

```

    </cache-listener>
    </region-attributes>
  </region>
</cache>
```



**Note:** The tutorial's sample XML files reference a cache DTD file from an earlier GemFire version. All GemFire schemas from previous GemFire versions and the current GemFire version are included with the product.

## 2. Start two servers using this persistent\_server.xml.

Open a terminal, and change directories to the tutorial directory,

<product\_directory>/SampleCode/tutorial. For example:

```
$ cd <product_directory>/SampleCode/tutorial
```

where <product\_directory> corresponds to the location where you installed vFabric GemFire.

Type the following command at the prompt:

```
$ gfsh start server --name=server1 --locators=localhost[55221]
--cache-xml-file=../xml/persistent_server.xml --server-port=0
```

This command will automatically create a working directory for this server named server1 under the <product\_directory>/SampleCode/tutorial directory where you executed the command.

Start a second server:

```
$ gfsh start server --name=server2 --locators=localhost[55221]
--cache-xml-file=../xml/persistent_server.xml --server-port=0
```

This command will automatically create a working directory for this server named server1 under the <product\_directory>/SampleCode/tutorial directory where you executed the command.

## 3. Add some posts from the client.

```

> post Isabella Hello
> post Ethan Goodbye
> posts
Ethan: Goodbye
Isabella: Hello
```

## 4. Kill both servers and restart them to make sure the posts are recovered.

```
$ gfsh stop server --dir=server1
$ gfsh stop server --dir=server2
```

When you restart persistent members, you call **gfsh start server** in parallel for each server.

The reason is that GemFire ensures that your complete data set is recovered before allowing the JVMs to start. Remember that each member is persisting only part of the Posts region. Each GemFire member waits until all posts are available before starting. This protects you from seeing an incomplete view of the posts region.

### Unix

```

$ gfsh start server --name=server1 --locators=localhost[55221] \
--cache-xml-file=xml/persistent_server.xml --server-port=0 \
& gfsh start server --name=server2 --locators=localhost[55221] \
--cache-xml-file=xml/persistent_server.xml --server-port=0 &
```

### Windows

```

prompt> start /B gfsh start server --name=server1
--locators=localhost[55221] ^
```

```
--cache-xml-file=xml/persistent_server.xml --server-port=0
prompt> start /B gfsh start server --name=server2
--locators=localhost[55221] ^
--cache-xml-file=xml/persistent_server.xml --server-port=0
```

### 5. View the posts from the client.

```
> posts
Ethan: Goodbye
Isabella: Hello
```

## Exploring Other Features with the Tutorial

Use the tutorial application to explore additional GemFire features.

### Faster and More Flexible Serialization with GemFire Serialization

In the tutorial, The PostID and Profile classes implemented `java.io.Serializable`. Java serialization works, but a few things about it are inefficient. Serialization uses reflection to find the fields to serialize. Serializing an object writes the entire classname and field names to the output stream. GemFire provides two serialization mechanisms that you can use to improve the performance and flexibility of serialization in your application. In addition, using GemFire serialization makes it easier to share data between Java, C++, and C# clients. For more information, see [Data Serialization](#) on page 257.

### Locator Redundancy

In the tutorial, we only used a single locator, which is a single point of failure for the application. In a production scenario you should use at least two locators. The locators property accepts a comma-separated list of locators.

### Executing Queries

Suppose you want to show all people who have someone listed as a friend. GemFire supports querying region contents using Object Query Language (OQL). See [Querying](#) on page 325 and the Javadocs for the `com.gemstone.gemfire.cache.query` package for more information. To find all people who have listed Ethan as a friend:

```
select p.key from /people.entrySet p, p.value.friends f where f='Ethan'
```

### Continuous Queries

In the tutorial, the client registered interest in updates to the people region by using a regular expression. You can also register for updates with an OQL query. For example, you can register interest in all of the posts from a particular user. For more information, see [Continuous Querying](#) on page 375.

### Partitioning Your Data Into Logical Groups for Faster Access

By default, posts are assigned to different peers based on the hash code of the key. That means no organization exists that determines which posts go to which servers. If you need to do work with all of the posts from a particular user, such as running a spelling check on them, you would have to access posts on many different peers. To make this type of operation efficient, it makes sense to group those posts by user name, so that the spellcheck could be performed in a single member. GemFire lets you do this with a PartitionResolver. The partition resolver lets you return a value that indicates the logical group that a key belongs to. In this sample application, the PartitionResolver could just return the author field of the PostID. That would tell GemFire to put all posts by the same author in the same member.

Once the posts are grouped together logically, you execute your spelling check on the member that actually stores the posts. GemFire lets you ship functions to a subset of the peers based on what keys those functions

intend to work with. The function is then executed in parallel on the subset of peers that host those keys. To execute the spelling check function:

```
SpellingCheck spellcheck = new SpellingCheck(); //implements Function
Set<String> authors = new HashSet<String>();
authors.add("Ethan");
FunctionService.onRegion(people).withFilter(authors).execute(spellcheck);
```

For more information on function execution, see [Function Execution](#) on page 417.

## Multiple Geographical Locations and WAN Distribution

If you have multiple data centers in remote locations, the synchronous replication between GemFire peers may cause too much latency if those peers are in different locations. GemFire supports gateway sender and gateway receiver configurations, which allow two or more remote sites to send updates asynchronously across a WAN. With a parallel gateway sender configuration, many updates are sent at once to improve throughput between sites. For more information on multi-site configurations, see [Multi-site \(WAN\) Configuration](#) on page 189.

## Cache Writers and Loaders

As you have seen, a CacheListener callback is invoked after an entry is changed. A CacheWriter callback is invoked before the entry is updated. You can block the update, or send the update to another system in a cache writer. You can add a CacheLoader callback to a region to fetch or generate a value if it was not present in the cache when a get was invoked. For more information on these callbacks, see [Events and Event Handling](#) on page 275.

## Eviction and Expiration

GemFire provides full support for the eviction and expiration features you might expect from a cache. You can limit the size of a region to a certain number of entries or a certain size in bytes. You can configure regions to expire entries after a certain amount of time, or simply evict entries when your heap is getting full.

Rather than lose the values completely, you can configure the region to overflow values onto disk.

See [Eviction](#) on page 248 and [Expiration](#) on page 250 for more information.



# Chapter 14

---

## Hello World Example

Walk through a simple set of GemFire configuration files and application code.

Each example:

- Initializes its cache and data regions using a declarative XML configuration file. The cache creation automatically creates the distributed system connection with any running members.
- Does its work.
- Closes its cache and exits.
- Contains a list of related Javadocs.

### Introduction to the Hello World Example

The Hello World example gives you a simple demonstration of GemFire capabilities.

#### What This Example Does

This example shows data distribution between two VMware® vFabric™ GemFire® Java applications.

- Both applications create a GemFire cache with a GemFire data region in it. Data regions hold and manage data entries, written in key/value pairs.
- The data region in both applications is configured to receive all data written to the region from anywhere in the GemFire system.
- One application is a producer, writing data to the cache, and the other is a consumer, receiving what the producer writes.
- Both applications report their caching activities to your screen. The producer reports on its writing activities. Both applications run a listener that reports on events in the application's cache.

#### Running the Hello World Example

Make sure your environment is configured to run the example.

#### Prerequisites

- [Installing vFabric GemFire](#) on page 27
- [Setting Up the Product Examples](#) on page 61

#### Procedure

1. Start two terminal sessions with their environments set up for GemFire and the product examples.
2. In both sessions, change directory (cd) to the helloworld directory under your examples installation.

3. In one terminal session, start the Consumer application.

```
java helloworld.HelloWorldConsumer
```

The application starts, creates the cache and region and then asks you to start the producer.

This example shows how a distributed region works with replication enabled. I'll create a replicate region, then the producer will create the same region and put entries into it. Because my region is a replicate, all of the producer's puts are automatically pushed into my region.

Connecting to the distributed system and creating the cache.  
Example region, /exampleRegion, created in cache.

Please start the HelloWorldProducer.

4. In the other terminal session, start the Producer application.

```
java helloworld.HelloWorldProducer
```

The application starts, creates the cache and region, puts data entry key/values pairs into its cache, and then exits.

Connecting to the distributed system and creating the cache.

Example region, /exampleRegion, created in cache.

Putting entry: Hello, World

Received afterCreate event for entry: Hello, World

Putting entry: Hello, Moon!

Received afterUpdate event for entry: Hello, Moon!

Closing the cache and disconnecting.

Please press Enter in the HelloWorldConsumer.



**Note:** The entry key is "Hello" for both puts, but the value changes. For each entry the producer puts, its listener receives the event and reports to your screen. The first put results in an entry creation. The second put results in an entry update, since the key, "Hello", is already in the cache.

5. Look at the consumer session.

Only the producer makes changes to its cache, but because of how the data region is configured, GemFire automatically distributes the changes to the consumer, so both caches receive the same events and end up with the same data in their region.

```
Received afterCreate event for entry: Hello, World  
Received afterUpdate event for entry: Hello, Moon!
```

## Walkthrough of the Example Files

The Hello World example uses standard GemFire configuration and application files.

You can find these files under the helloworld directory in your examples installation.

### gemfire.properties

gemfire.properties is the vFabric GemFire system configuration file. Use it to define how processes in a GemFire system (in the Hello World example, two Java applications) find each other and communicate.

Also use the file to configure logging, security, statistics gathering, and more. Each member has its own gemfire.properties file.

The file used in this example sets non-default values for these properties:

- Multicast port location used by the applications to establish communication:

```
mcast-port=10334
```

- Level of logging the system should provide:

```
log-level=warning
```

### **xml/HelloWorld.xml (cache.xml)**

The xml/HelloWorld.xml file is the declarative XML configuration for the vFabric GemFire cache. Use this file to set up general cache facilities and behavior and to create and initialize cached data regions.

Although it can have any name, this file is generally referred to as `cache.xml`.

#### **Contents of HelloWorld.xml**

- XML version and DOCTYPE declarations (required).
- `<cache>` element. Highest level element for peer and server caches. These are peer caches. (Client caches are defined by `<client-cache>`.)
- `<region>` element. Defines the data region used for the example and specifies all non-default region configuration settings.
  - This region is configured as a replicate, meaning that any data either application writes into it is sent automatically to the other application's cache.
  - The region also has a listener installed named SimpleCacheListener.

```
<?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>
  <region name="exampleRegion" refid="REPLICATE">
    <cache-listener>
      <class-name>helloworld.SimpleCacheListener</class-name>
    </cache-listener>
  </region>
</cache>
```



**Note:** The tutorial's sample XML files reference a cache DTD file from an earlier GemFire version. All GemFire schemas from previous GemFire versions and the current GemFire version are included with the product.

### **HelloWorldProducer.java and helloworld/HelloWorldConsumer.java**

These example programs are Java applications that use the vFabric GemFire distributed caching API to connect to other applications and share data in memory.

The files are in the GemFire helloworld examples package of classes. They both use the GemFire Cache, CacheFactory, and Region APIs.

#### **Example Programs Common Package and Import Declarations**

In addition to the packages shown, the consumer program imports a couple of Java APIs for detecting keystrokes.

```
package helloworld;

import com.gemstone.gemfire.cache.Cache;
```

```
import com.gemstone.gemfire.cache.CacheFactory;
import com.gemstone.gemfire.cache.Region;
```

### Creating Caches and Connecting to Each Other

The HelloWordProducer and HelloWordConsumer programs create their cache, specifying the XML file to use. The cache creation automatically creates the distributed system using the gemfire.properties file settings. The distributed system is what connects the two programs to each other.

```
Cache cache = new CacheFactory()
    .set("cache-xml-file", "xml/HelloWorld.xml")
    .create();
```

### Getting the Data Region

The programs get the example region that was declared in the XML file.

```
Region<String, String> exampleRegion = cache.getRegion("exampleRegion");
```

### Working in the Cache

The producer puts data into the region and reports its activities to standard output.

```
System.out.println("Putting entry: Hello, World");
exampleRegion.put("Hello", "World");
System.out.println("Putting entry: Hello, Moon!");
exampleRegion.put("Hello", "Moon!");
```

Because of how their data regions are configured in the XML file, the data put by the producer is automatically pushed to the consumer's cache.

The consumer application code does nothing with the data, but its region's listener handles the data events.

### Closing the Cache and Disconnecting

```
cache.close();
```

## helloworld/SimpleCacheListener.java

SimpleCacheListener.java is a Java application plug-in written to handle cache region events.

You can code application plug-ins to respond to a variety of events in the GemFire system. This listener is installed on the region in both applications, through the XML file declarations.

### SimpleCacheListener, afterCreate, and afterUpdate Callbacks

The following code is called when the entry is put into the cache.

```
public void afterCreate(EntryEvent<K,V> e) {
    System.out.println("    Received afterCreate event for entry: " +
        e.getKey() + ", " + e.getNewValue());
}
public void afterUpdate(EntryEvent<K,V> e) {
    System.out.println("    Received afterUpdate event for entry: " +
        e.getKey() + ", " + e.getNewValue());
}
```

# Chapter 15

---

## QuickStart Examples

The examples each demonstrate a specific vFabric GemFire feature. They also show system member startup and shutdown, and provide a list of related Javadocs.

Each example:

- Initializes its cache and data regions using a declarative XML configuration file. The cache creation automatically creates the distributed system connection with any running members.
- Does its work, closes its cache, and exits.

### Setting Up Your Environment

To start using the examples, start terminal sessions with environments set for GemFire and the examples, then locate the example files.

If you have not already done so, install and configure vFabric GemFire and the GemFire product examples. See the following topics:

- [Installing vFabric GemFire](#) on page 27
- [Setting Up the Product Examples](#) on page 61

To begin setting up your environment, perform the following:

1. Start two terminal sessions with their environments set up for GemFire and the product examples.
2. In both sessions, change directory (cd) to the quickstart directory under `<product_install_dir>/SampleCode` installation.

If you have problems running the examples, see the [Troubleshooting Checklist](#) on page 110.

### Example Files

Source and configuration files are located under the examples installation quickstart directory:

- `gemfire.properties`, used by all of the examples
- Java source files in the quickstart subdirectory
- XML cache configuration files in the `xml` subdirectory

For descriptions and example walkthroughs for these different file types, see [Walkthrough of the Example Files](#) on page 82.

### Management and Monitoring Example

This example uses an MBean server to manage and monitor a node in a distributed system. One node acts as an MBean server that gets information from other nodes.

## Running the Example



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

1. Change to the <Install\_Directory>/SampleCode/quickstart directory.

2. Open two terminal sessions and add the following files to the CLASSPATH:

- tomcat-embed-logging-juli.jar
- tomcat-embed-jasper.jar
- tomcat-embed-core.jar

3. In a terminal session, run the following command to start the managed node:

```
java quickstart.ManagedNode
```

The managed node retrieves an MBean locally from the management service, creates a local cache and the exampleRegion region, adds information retrieved from the MBean such as entry count to the region, and displays information about the items. Wait for the following message before continuing:

```
Start Manager Node and wait till it completes
```

4. Open a second terminal session and run the following command:

```
java quickstart.ManagerNode
```

The manager node creates a region and retrieves the region MBean from the managed node you started in the first session. Wait for the following message before continuing:

```
Press enter in Managed Node
```

5. In the first session, press ENTER.

## Example Source Files

**Table 5: Cache configuration files, located in SampleCode/quickstart/xml**

managed-node.xml	Creates a cache and region for MBean data.
------------------	--

**Table 6: Java program files, located in SampleCode/quickstart/quickstart**

ManagerNode.java	Retrieves the region MBean for all members of a GemFire distributed system.
ManagedNode.java	Retrieves a local region MBean.

### Related Topics

[GemFire Management and Monitoring](#) on page 429

## Related Javadocs

- [com.gemstone.gemfire.cache.CacheFactory](#)
- [com.gemstone.gemfire.management.ManagementService](#)
- [com.gemstone.gemfire.management.RegionMXBean](#)

## JSON Example

In this example you store a JSON document in a GemFire cache and query the document.

## Running the Example



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

1. Open a terminal session and switch to the <Install\_directory>/SampleCode/quickstart directory.

2. Run the following command:

```
gfsh start server --name=server_json --cache-xml-file=xml/Server.xml
```

3. Wait for the server to start.

4. Open a second terminal session and switch to the <Install\_directory>/SampleCode/quickstart directory.

5. Add the following jar file to the CLASSPATH environment variable:

```
<Install_directory>/lib/jackson-core-asl-1.9.9.jar.
```

6. Run the following command:

```
java quickstart.JSONClient
```

The client displays the following JSON document:

```
{
    "firstName" : "John",
    "lastName" : "Smith",
    "age" : 25,
    "address" : {
        "streetAddress" : "21 2nd Street",
        "city" : "New York",
        "state" : "NY",
        "postalCode" : "10021"
    },
    "phoneNumber" : [ {
        "type" : "home",
        "number" : "212 555-1234"
    }, {
        "type" : "fax",
        "number" : "646 555-4567"
    } ]
}
```

The client retrieves the JSON document, queries it, and displays the results.

7. Switch to the first session and stop the cache server using the following command:

```
gfsh stop server --dir=server_json
```

## Example Source Files

**Table 7: Cache configuration files, located in SampleCode/quickstart/xml**

Server.xml	Configures a cache to serve caching clients. The example region also is configured with a loader.
JsonClient.xml	Configures a region as a client region in a client/server cache. The region's pool connects to a cacheserver listening on port 40404.

**Table 8: Java program files, located in SampleCode/quickstart/quickstart**

JSONClient.java	Creates a client that stores a JSON document in the cache and then queries the document.
SimpleCacheLoader.java	A very simple CacheLoader implementation
<b>Related Topics</b>	
<a href="#">Adding JSON Documents to the GemFire Cache</a> on page 270	

**Related Javadocs**

- [com.gemstone.gemfire.pdx.JSONFormatter](#)
- [com.gemstone.gemfire.pdx.PdxInstance](#)

**Replicated Caching Example**

Replicated caching gives you a complete copy of your data in each member in the distributed system that hosts the replicated region.

Any change made by any distributed system member is automatically forwarded into the replicated region. You can use this to keep a complete copy in one place for such things as high availability and synchronization with an outside data store. In a client/server installation, backup servers that replicate primary server data can take over when the primary fails, with no data loss. Read more about replicated regions in [Distributed and Replicated Regions](#) on page 233.

**Running the Example**

This example shows a producer/consumer system. The producer acts as a data feed, putting data into the region using a replicated proxy. A replicated proxy does not store anything in its own cache but sends it to any replicated regions it finds in its distributed system.

The consumer region is configured as replicated so anything the producer adds or updates is automatically copied to the consumer. The consumer region has a listener that reports all updates to standard out, so you can see when the copies arrive. A GemFire cache listener handles changes to the data region.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

1. In one terminal session, start the consumer:

```
$ java quickstart.PushConsumer
```

2. After that session starts, in another session, start the producer:

```
$ java quickstart.PushProducer
```

**Example Source Files**

Program and cache configuration source files for the producer and consumer, including the listener, SimpleCacheListener.java, declared in the PushConsumer.xml file.

**Table 9: Cache configuration files, located in SampleCode/quickstart/xml**

PushProducer.xml	Configures a producer region for a data replication example.
PushConsumer.xml	Configures a consumer region for a data replication example.

**Table 10: Java program files, located in SampleCode/quickstart/quickstart**

PushProducer.java	Puts data into the region using a replicated proxy.
PushConsumer.java	Automatically copies producer updates.
SimpleCacheListener.java	A CacheListener that reports consumer cache events.

### Related Javadocs

See also:

- `com.gemstone.gemfire.cache.CacheFactory`
- `com.gemstone.gemfire.cache.Cache`
- `com.gemstone.gemfire.cache.Region`
- REPLICATE and REPLICATE\_PROXY in `com.gemstone.gemfire.cache.RegionShortcut`

## Server Managed Caching Example

This example uses the client/server topology to scale an application vertically, with many clients benefitting from centralized server-side data management.

Server-managed caching is the model for independent scalability of data and application tiers. You isolate data management in cache servers and connect to them from application server clients, so many clients benefit from the management resources of the servers.

GemFire's client/server caching gives you dynamic server pool management, server load balancing and conditioning, logical server grouping, and automatic server failover for high availability.

The servers run in one distributed system. Each client runs in its own separate distributed system. The clients forward data requests to the servers that their caches cannot fulfill. The clients also forward client-side updates. Additionally, clients can register interest in server-side data, either by specifying the data keys or by registering queries that run continuously. In both cases, updates on the server side are automatically forwarded to the client.

### Running the Example

This example shows a simple client/server installation with one worker client, that gets and puts data in its cache, and one consumer client, that just receives events from the server. Both clients are connected to a single cache server. The worker client requests data from the server. If the server does not have the data cached, it automatically runs a data loader to get it. The server returns the data to the worker client. The worker client then updates the data in its cache. The update is automatically distributed to the server. The server forwards all changes made by the worker client to the consumer client. Both clients have an asynchronous listener that reports local caching activity to standard out, so you can see what is happening.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

1. Change to the <InstallDirectory>/SampleCode/quickstart directory.
2. In one session, start the cache server (see [vFabric GemFire Servers](#) on page 741 for more information):

```
$ gfsh start server --name=server_mc --cache-xml-file=xml/Server.xml
```

The gfsh command starts a cache server in the background, displays the status of the server and some environment information, and then exits to the command prompt.

3. In the same session, start the consumer client:

```
$ java quickstart.ClientConsumer all-keys
```

- When the consumer client asks you to start the worker client, start it in another session:

```
$ java quickstart.ClientWorker
```

- Follow the instructions on the screens, noting the listener output from each client. When both clients have exited, stop the cache server:

```
$ gfsh stop server --dir=server_mc
```

You can also use the deprecated `cacheserver` command to stop the cache server:

```
$ cacheserver stop
```

### Example Source Files

Program and cache configuration files for the clients and the server, including the loader and listener declared in the **Server.xml** and **Client.xml** files. (The server is a GemFire cacheserver process and does not have a Java source file.)

**Table 11: Cache configuration files, located in SampleCode/quickstart/xml**

Server.xml	Configures a cache to serve caching clients. The example region also is configured with a loader.
Client.xml	Configures a region as a client region in a client/server cache. The region's pool connects to the cacheserver.

**Table 12: Java program files, located in SampleCode/quickstart/quickstart**

ClientWorker.java	A client program that exercises the server cache.
ClientConsumer.java	A consumer client that registers interest in events on the server. The server sends automatic updates for the events.
SimpleCacheLoader.java	A very simple CacheLoader implementation.
SimpleCacheListener.java	A CacheListener that reports cache events.

#### Related Topics

[Topology and Communication General Concepts](#) on page 163

[Client/Server Configuration](#) on page 177

#### Related Javadocs

- [com.gemstone.gemfire.cache.client.ClientCacheFactory](#)
- [com.gemstone.gemfire.cache.client.Pool](#)
- [REPLICATE](#) in [com.gemstone.gemfire.cache.RegionShortcut](#)
- [CACHING\\_PROXY](#) in [com.gemstone.gemfire.cache.client.ClientRegionShortcut](#)

### Partitioned Data Caching Example

If a data set is too large for one member to hold, you can store it in more than one member and work with it as if it were a single entity.

You set up a partitioned region by creating region instances with the same name in all members that will hold the data, and enable partitioning during region creation. GemFire handles the physical location of data in the members that participate in a partitioned region.

You can configure your partitioned region to store redundant copies of each member's data in other members where the region is defined. This way, every data entry is hosted by multiple members, giving the region seamless high availability in the case of a member failure.

## Running the Example

In this example, two members have the same partitioned region configured, with one extra copy of data entries. The example creates three entries in the partitioned region and reads them, then destroys an entry, invalidates an entry, and reads them all again.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

1. In one session, start the first member:

```
$ java quickstart.PartitionedRegionVM1
```

2. When the first member tells you to do so, in the other session, start the second member:

```
$ java quickstart.PartitionedRegionVM2
```

This example deliberately compromises redundancy by closing PartitionedRegionVM2. Because of this, you will see this warning from PartitionedRegionVM1:

```
[warning 2008/09/08 11:34:41.587 PDT PartitionedRegion /PartitionedRegion
    Message Processor1> tid=0x2b]
Redundancy has dropped below 1 configured copy to 0 actual copies for
/PartitionedRegion
```

## Example Source Files

Program and cache configuration source files for the partitioned region shared by JVMs 1 and 2, which use the same cache XML configuration file.

**Table 13: Cache configuration files, located in SampleCode/quickstart/xml**

PartitionedRegion.xml	Configures the partitioned region.
-----------------------	------------------------------------

**Table 14: Java program files, located in SampleCode/quickstart/quickstart**

PartitionedRegionVM1.java	The initial member, which creates entries in the partitioned region then destroys them.
PartitionedRegionVM2.java	The second member, which reads entries from the partitioned region.

## Related Topics

[Distributed System and Cache Configuration](#) on page 133

[Partitioned Regions](#) on page 213

## Related Javadocs

See also:

- [com.gemstone.gemfire.cache.CacheFactory](#)
- [com.gemstone.gemfire.cache.Region](#)
- [com.gemstone.gemfire.cache.PartitionAttributes](#)

- com.gemstone.gemfire.cache.PartitionAttributesFactory
- PARTITION\_REDUNDANT in com.gemstone.gemfire.cache.RegionShortcut
- PartitionAttributes in com.gemstone.gemfire.cache.RegionFactory

## Fixed Partitioning

You can set up partitioned regions for specific data that must reside on specific physical computers, or when your data must be close to certain hardware components.

You implement fixed partitioning by configuring standard partitioning and then specifying the exact member where each data entry resides. You assign the data entry to a bucket and partition, and name specific members as primary and secondary hosts of each partition.

This configuration gives you complete control over the locations of your primary and any secondary buckets for the region.

### Running the Example

This example shows basic region operations on a fixed-partitioned region, which has data partitions configured on two JVMs, Peer1 and Peer2. The partitions are configured as follows:

Peer1 (defined by `FixedPartitionPeer1.xml`) configures partition Q1 as the primary and partition Q2 as the secondary.

Peer2 (defined by `FixedPartitionPeer2.xml`) configures partition Q2 as the primary and partition Q1 as the secondary.

The region is configured to create one extra copy of each data entry. The copies are placed on different JVMs. When one JVM shuts down, no data is lost and operations can proceed normally on the other JVM.

Fixed partition "Q1" is associated with data of following months: Jan, Feb, Mar, Apr, May, Jun.

Fixed partition "Q2" is associated with data of following months: Jul, Aug, Sep, Oct, Nov, Dec .



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

1. Change to the `<Install_Directory>/SampleCode/quickstart` directory.
2. In a terminal session, start Peer1:  

```
$ java quickstart.FixedPartitionPeer1
```
3. In a second terminal session, start Peer2:  

```
$ java quickstart.FixedPartitionPeer2
```

### Example Source Files

**Table 15: Cache configuration files, located in `SampleCode/quickstart/xml`**

FixedPartitionPeer1.xml	Configures fixed partitioning attributes for a partitioned region on Peer1.
FixedPartitionPeer2.xml	Configures fixed partitioning attributes for a partitioned region on Peer2.

**Table 16: Java program files, located in SampleCode/quickstart/quickstart**

FixedPartitionPeer1.java	Defines a partitioned region that has a primary partition named "Q1" with 6 buckets.
FixedPartitionPeer2.java	Defines a partitioned region that has a primary partition named "Q2" with 6 buckets.
QuarterPartitionResolver.java	This custom resolver maps a user-defined partition to a data-member node.

**Related Topics**[Partitioned Regions](#) on page 213**Related Javadocs**

- [com.gemstone.gemfire.cache.FixedPartitionResolver](#)
- [com.gemstone.gemfire.cache.FixedPartitionAttributes](#)

**Reliable Event Notification Examples**

These examples demonstrate how you can fine-tune the events your clients receive from the server by using event subscriptions.

You can register subscriptions for specific data keys, data keys matching a regular expression, and data matching a continuously running query. The client receives automated notifications from the server whenever the indicated data changes. For key registration, the change is any change to the data object. For continuous queries, the change is any change to the query result set for the query.

**Interest Registration Example**

Client interest registration can give you automatic updates from the server for individual keys, key lists, and all keys matching a regular expression.

This example modifies the Server Managed Caching Example's default interest registration to show explicit key registration and the use of regular expressions.

**Running the Example**

The default behavior for the [Server Managed Caching Example](#) on page 89 is to register interest in all data in the region.

The example has one worker client, which gets and puts data in its cache, and one consumer client, which receives events from the single cacheserver. The consumer client registers interest in some or all keys in the data region. The worker client updates its cache, and the updates are automatically forwarded to the server cache. The server forwards to the consumer only those events in which the consumer has registered interest. The clients have an asynchronous listener that reports local cache activity to standard out, so you can see what is happening. The server uses a cache loader to load data it does not already have in its cache when the worker client requests it.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

You will run this example twice, once registering interest in a specific set of keys, the second time, registering interest in keys matching a regular expression.

1. In one session, start the cache server:

```
$ gfsh start server --name=server_int_reg --cache-xml-file=xml/Server.xml
```

The gfsh command starts a cache server in the background, displays the status of the server and some environment information, and then exits to the command prompt.

2. In the same session, start the consumer client:

```
$ java quickstart.ClientConsumer keyset
```

3. When the consumer client asks you to do so, start the worker client in another session:

```
$ java quickstart.ClientWorker
```

4. Follow the instructions on the screens, noting the listener output from each client. After both clients exit, stop the cacheserver:

```
$ gfsh stop server --dir=server_int_reg
```

You can also use the deprecated cacheserver command:

```
$ cacheserver stop
```

5. To run the example using a regular expression for interest registration, run the example again, replacing "keyset" in the ClientConsumer startup with "regex".

## Example Source Files

Program and cache configuration source files for the clients and the server, including the loader and listener declared in the Server.xml and Client.xml files, respectively. (The server is a GemFire cacheserver process and does not have an example source file.)

**Table 17: Cache configuration files, located in SampleCode/quickstart/xml**

Server.xml	Configures a cache to serve caching clients. The example region also is configured with a loader.
Client.xml	Configures a region as a client region in a client/server cache. The region's pool connects to the cacheserver.

**Table 18: Java program files, located in SampleCode/quickstart/quickstart**

ClientWorker.java	A client program that exercises the server cache.
ClientConsumer.java	A consumer client that registers interest in events on the server. The server sends automatic updates for the events.
SimpleCacheLoader.java	A very simple CacheLoader implementation.
SimpleCacheListener.java	A CacheListener that reports cache events.

### Related Topics

[Client/Server Configuration](#) on page 177

[Client-to-Server Event Distribution](#) on page 279

### Related Javadocs

See also:

- [com.gemfire.cache.Region.registerInterest](#)

## Continuous Querying Example

The client registers a query with the server and chooses whether to receive an initial query result set. After initialization, the query runs continuously against server cache changes, and the client receives events for any changes to the query result set.

Here's a typical use case: A client application displays active stock trades to a screen. The client creates a CQ on the server's trading data set with a WHERE clause specifying "status='active'". The client requests an initial result set for the CQ and initializes its screen display with those results. From then on, the client updates its screen display with the CQ events as they arrive from the server. An entry whose status goes from 'active' to 'inactive' on the server is reported to the client as a query destroy event for the entry, and the client application removes the entry from the active trades display. Newly 'active' entries are reported as additions to the result set and are added to the display.

## Running the Example

This example has one client that connects to a cache server, creates a continuous query, and then updates the server with data that changes the query result set. The client reports the updates to standard out. It also has a CQ listener that reports all CQ events, so you can see the effects on the result set. The server uses a cache loader to load any data requested by the client that it does not already have in its cache.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

1. In one session, start the cache server:

```
$ gfsh start server --name=server_cq --cache-xml-file=xml/CqServer.xml
```

The gfsh command starts a cache server in the background, displays the status of the server and some environment information, and then exits to the command prompt.

2. In the same session, start the CQ client:

```
$ java quickstart.CqClient
```

The client runs a CQ on the server, then creates cache events that modify the CQ result set, prompting the server to send it CQ events.

3. When the client exits, stop the cache server:

```
$ gfsh stop server --dir=server_cq
```

You can also use the deprecated cacheserver command to stop the cache server:

```
$ cacheserver stop
```

## Example Source Files

Program and cache configuration source files for the client and server, including the loader and listener declared in **CqServer.xml**. (The server is a GemFire cacheserver process and does not have an example source file.)

**Table 19: Cache configuration files, located in SampleCode/quickstart/xml**

CqServer.xml	Configures a cache to serve CQ clients.
CqClient.xml	Configures a region as a client region in a client/server cache. The region's pool connects to the cacheserver.

**Table 20: Java program files, located in SampleCode/quickstart/quickstart**

CqClient.java	A client that creates and executes continuous queries on the server and waits for the events.
SimpleCacheLoader.java	A very simple CacheLoader implementation.
SimpleCacheListener.java	A CacheListener that reports cache events.
SimpleCqListener.java	A CqListener that reports continuous query events.

**Related Topics**[Delta Propagation](#) on page 317**Related Javadocs**

See also:

- [com.gemstone.gemfire.cache.query.QueryService](#)
- [com.gemstone.gemfire.cache.query.CqQuery](#)
- [com.gemstone.gemfire.cache.query.CqListener](#)
- [com.gemstone.gemfire.cache.query.CqEvent](#)

**Durable Event Messaging Example**

With durable messaging configured for a client, servers save specified events in queue while the client is disconnected and then play them back when the client reconnects.

You can select which continuous queries or interest registrations are durable, so the server only queues events you really need to retain if your client goes down.

**Running the Example**

This example runs a server that updates its cache. It has a durable client that registers durable interest in some keys and non-durable interest in others. You start the client, stop it, and start it again while the server is updating the cache, so you can see that the durable events are saved by the server and played back when the client reconnects. The client has an asynchronous listener that reports local cache activity to standard out, so you can see what is going on.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

1. In one session, start the server:

```
$ java quickstart.DurableServer
```

2. In another session, start the durable client:

```
$ java quickstart.DurableClient
```

3. After the client starts, press Enter in the server session once or twice to see the events arrive at the client side.
4. Press Enter in the client session to exit the client application.
5. When the client is stopped, in the server session, press Enter twice to create new events.
6. Restart the client application.

You see the events that the server stored for the durable interest played back to the client.

7. Press Enter to exit the client again.

8. Type `exit` and press Enter to exit the server.

### Example Source Files

Program and cache configuration source files for the client and server, including the listener declared in the client and server xml files.

**Table 21: Cache configuration files, located in SampleCode/quickstart/xml**

DurableServer.xml	Configures a cache to serve Durable caching clients. The example region also is configured with a loader and listener.
DurableClient.xml	Configures a region as a client region in a Durable cache.

**Table 22: Java program files, located in SampleCode/quickstart/quickstart**

DurableServer.java	The server to which the DurableClient connects
DurableClient.java	The client that connects to the DurableServer.java.
SimpleCacheListener.java	A CacheListener that reports cache events.

Related Topics
<a href="#">How Events Work</a> on page 275
<a href="#">Implementing Durable Client/Server Messaging</a> on page 301

### Related Javadocs

See also:

- durable-client-id and durable-client-timeout in `com.gemstone.gemfire.distributed.DistributedSystem`
- readyForEvents and close methods in `com.gemstone.gemfire.cache.client.ClientCache`
- methods for registering and unregistering interest in `com.gemstone.gemfire.cache.Region`

## Persisting Data to Disk Example

Persistence creates a complete backup of the region data on disk.

At region creation time, if the disk files are already present, the data is recovered from disk and used to initialize the region.

### Running the Example

In this example, region data is persisted to the subdirectory `persistData1`. You run the example twice so you can see how the persisted data is loaded into the region on the second run.

 **Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

1. In a single session, run the example:

```
$ java quickstart.DataPersistence
```

No data is in the region on the first run, so the get operation returns null and the program reports that no entry was found. Then the program puts the entry with value, 'value1'.

2. When the program exits, look in the persistData1 directory. You will see the data files used for persistence.
3. Run the program again with the same command.

The second time, at creation, the region is initialized from disk, so the get operation returns 'value1'.

### Example Source Files

Program and cache configuration source files:

**Table 23: Cache configuration files, located in SampleCode/quickstart/xml**

DataPersistence.xml	Configures a region to persist its data to disk. The data files are written to the directories specified in the disk-dirs elements.
---------------------	---

**Table 24: Java program files, located in SampleCode//quickstart/quickstart**

DataPersistence.java	Demonstrates how to create persistent data and how the region is initialized from persistent data, if it exists.
----------------------	--

#### Related Topics

[Persistence and Overflow](#) on page 245

### Related Javadocs

See also:

- [com.gemstone.gemfire.cache.DiskStoreFactory](#)
- [com.gemstone.gemfire.cache.DiskStore](#)
- options with PERSISTENT in [com.gemstone.gemfire.cache.RegionShortcut](#)

### Overflowing Data to Disk Example

Use disk with in-memory caching to overflow data to disk while your region is larger than a configured size. Overflow management is transparent to your application; the disk files are treated as an extension of the region in memory.

Disk data is maintained like in-memory data, and entry requests are automatically retrieved from the disk files as needed. This is a great alternative to eviction for applications with both a large volume of data and a high cost associated with accessing the original, outside data source.

**Note:** Overflow does not provide a backup of your data; it provides a disk extension to your in-memory storage space for the region. If you want to back up your data, see the Persisting Data to Disk Example.

### Running the Example

In this example overflow data is written to the subdirectory overflowData1. Notice that the disk files are only there while the region is in memory. When the region is removed from memory, its overflow files are no longer needed, so they are destroyed.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

In a single session, run the example:

```
$ java quickstart.DataOverflow
```

### Example Source Files

Program and cache configuration source files:

**Table 25: Cache configuration files, located in SampleCode//quickstart/xml**

DataOverflow.xml	Configures a region to overflow to disk when the region reaches a certain capacity. The data files are written to the subdirectories specified in the disk-dirs elements.
------------------	---

**Table 26: Java program files, located in SampleCode//quickstart/quickstart**

DataOverflow.java	Shows cached data overflow to disk.
<b>Related Topics</b>	
<a href="#">Persistence and Overflow</a> on page 245	

### Related Javadocs

- [com.gemstone.gemfire.cache.EvictionAttributes](#)
- [com.gemstone.gemfire.cache.DiskStoreFactory](#)
- [com.gemstone.gemfire.cache.DiskStore](#)

## Cache Expiration Example

Expiration helps you ensure that your data is current and pertinent by removing old and unused entries from the cache.

It can also give you some control over cache size. When you try to access an expired entry, the value is not there and your cache looks to another source for a fresh update. Usually it calls the loader you have installed on the data region.

You can expire entries based on how recently they were added or updated, or based on how recently they were accessed by your application, added, or updated. Add/update-based expiration, called time-to-live or TTL, keeps your application from accessing stale data. Add/update/access-based, or idle-time, expiration keeps your application from accessing stale data and from using memory for data it isn't interested in accessing.

### Running the Example

This example uses idle time expiration, configured through the XML file to destroy expired entries. The program creates three entries and then waits beyond the expiration time. During the wait time, the program updates one entry and accesses another. When the expiration time elapses, only the third, unaccessed and unmodified entry is expired. The cache listener, installed on the region, reports on all changes to the entries.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

In a single session, run the example:

```
$ java quickstart.DataExpiration
```

### Example Source Files

Program and cache configuration source files for the example, including the listener declared in DataExpiration.xml:

**Table 27: Cache configuration files, located in SampleCode//quickstart/xml**

DataExpiration.xml	Configures a region with entry idle time expiration.
--------------------	--

**Table 28: Java program files, located in SampleCode/quickstart/quickstart**

DataExpiration.java	Adds data to the cache and lists entries before and after expiration.
SimpleCacheListener.java	A CacheListener that reports cache events.

### Related Topics

[Expiration](#) on page 250

### Related Javadocs

- Expiration methods on `com.gemstone.gemfire.cache.RegionFactory`

## Cache Eviction Example

Eviction controls your data region size by removing least recently used (LRU) entries to make way for new data.

You can use eviction to control how much heap your data regions use. It kicks in when your data region reaches a specified size or entry count. Size can be absolute or a percentage of your application's current heap.

### Running the Example

In this example, the data region is configured to keep the entry count at 10 or below by destroying LRU entries. A cache listener installed on the region reports the changes to the region entries.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

In a single session, run the example:

```
$ java quickstart.DataEviction
```

### Example Source Files

Program and cache configuration source files for the example, including the listener declared in the DataEviction.xml file:

**Table 29: Cache configuration files, located in SampleCode//quickstart/xml**

DataEviction.xml	Configures a region to destroy entries when the region reaches a certain capacity. Includes a listener to report on the activity.
------------------	---

**Table 30: Java program files, located in SampleCode//quickstart/quickstart**

DataEviction.java	Demonstrates data eviction by adding more entries than the cache is configured to hold.
SimpleCacheListener.java	A CacheListener that reports cache events.

**Related Topics**[Eviction](#) on page 248**Related Javadocs**

- [com.gemstone.gemfire.cache.EvictionAttributes](#)

**Querying Example (OQL)**

With OQL querying, you can combine data from multiple regions and drill down into nested region objects, extracting only the data points you need.

OQL is similar to SQL but it works with objects. You can also use indexing to optimize your query performance.

**Running the Example**

This example creates an example region with data and then runs a handful of queries against it. The first query selects everything from the region so you can see all data the queries are run against.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

In a single session, run the example:

```
$ java quickstart.Querying
```

**Example Source Files**

Program and cache configuration source files:

**Table 31: Cache configuration files, located in SampleCode/quickstart/xml**

Querying.xml	Initializes a region with sample Portfolio data for querying.
--------------	---

**Table 32: Java program files, located in SampleCode/quickstart/quickstart**

Querying.java	Demonstrates querying on a set of data in a region.
Portfolio.java	A stock portfolio that consists of multiple Position objects that represent shares of a stock.
Position.java	Represents shares of stock held in a portfolio.

**Related Topics**[Querying](#) on page 325**Related Javadocs**

- [com.gemstone.gemfire.cache.query](#)

## Transactions Example

You can group cache updates with vFabric GemFire transactions and create dependencies between cache modifications so that they all complete together or fail together.

You can execute transactions against the data in the distributed cache similar to the way you execute transactions in a database, with the standard begin, commit, and rollback operations. GemFire transactions are compatible with Java Transaction API (JTA) transactions and, when run inside a J2EE container, are also compatible with the container's JNDI and JTA transaction manager.

### Running the Example

This example runs and commits one transaction and then runs and rolls back another.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

In a single session, run the example:

```
$ java quickstart.Transactions
```

### Example Source Files

Program and cache configuration files:

**Table 33: Cache configuration files, located in SampleCode//quickstart/xml**

Transactions.xml	Configures a simple distributed region.
------------------	---

**Table 34: Java program files, located in SampleCode//quickstart/quickstart**

Transactions.java	Demonstrates executing transactions against a distributed cache.
-------------------	--

### Related Topics

[Transactions](#) on page 383

### Related Javadocs

- [com.gemstone.gemfire.cache.CacheTransactionManager](#)
- [com.gemstone.gemfire.cache.TransactionEvent](#)
- [com.gemstone.gemfire.cache.TransactionListener](#)

## Delta Propagation Example

Delta propagation lets you send deltas (changes) instead of full values when you propagate events from one cache to another.

In distributed data management systems, most data is created once and then updated frequently. The updates are sent to other members for event propagation, redundancy management, and cache consistency in general. With delta propagation, you track only the changes in an updated object and send only the parts that have changed: the deltas. This feature brings significant performance benefits because of lower network transmission and object serialization/deserialization costs. The cost savings can be significant, especially when changes to an object instance are relatively small compared to the overall size of the instance.

## Running the Example

The example shows a very simple client/server installation with one feeder client, that gets and updates data in its cache, and one receiver client, that receives data events from the server. Both are connected to a single cacheserver. The feeder client requests data from the server and then updates the data in its cache. The update to the feeder client's cache also updates the server's cache. The server automatically forwards these data events to the receiver client, so the receiver's cache is updated as well. The clients have an asynchronous listener that reports local cache activity to standard out.

The key to delta propagation is in the coding of the object stored in the region entry value. The entry value object, DeltaObj, used in this example implements GemFire's com.gemstone.gemfire.Delta, so that it sends and receives only the object deltas, rather than the full object. So, when the feeder client changes the object, the object tracks what is being changed. When the feeder client puts the object into its cache and GemFire distributes the put event to the server, the object provides only the delta for the distribution. The server receives the delta, uses the same object implementation to read the delta and update its cached copy, then forwards the delta to the receiver client, who does the same.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

1. In one session, start the GemFire cacheserver (see [vFabric GemFire Servers](#) on page 741 for more information):

```
$ gfsh start server --name=server_dp  
--cache-xml-file=../xml/DeltaServer.xml
```

The gfsh command starts a cache server in the background, displays the status of the server and some environment information, and then exits to the command prompt.

2. In the same session, start the feeder client:

```
$ java quickstart.DeltaPropagationClientFeeder
```

3. The feeder prompts you to press Enter. Before you do, in another session, start the receiver client:

```
$ java quickstart.DeltaPropagationClientReceiver
```

4. Go back to the feeder session and press Enter.

5. Follow the instructions on the screens, noting the listener output from each client.

6. Stop the receiver by pressing Enter.

7. Stop the cache server:

```
$ gfsh stop server --dir=server_dp
```

You can also use the deprecated cacheserver command to stop the cache server:

```
$ cacheserver stop
```

## Example Source Files

Program and cache configuration files for the clients and the server, including the loader and listener declared in the Server.xml and Client.xml files. (The server is a GemFire cacheserver process and does not have an example source file.)

**Table 35: Cache configuration files, located in SampleCode//quickstart/xml**

DeltaServer.xml	Initializes a cache to serve the /root/cs_region region, waiting for client communication.
DeltaClient1.xml	Initializes a client of a cache server that runs on port 40404. Loads values and sends updates to the server.

DeltaClient2.xml	Initializes a client of a cache server that runs on port 40450. Loads values and sends updates to the server.
------------------	---

**Table 36: Java program files, located in SampleCode//quickstart/quickstart**

DeltaObj.java	A sample Object class that implements Delta.
DeltaPropagationClientFeeder.java	The client that connects to the DeltaPropagationServer.java.
DeltaPropagationClientReceiver.java	Sets up a new cache with a ReceiverListener.
DeltaSimpleListener.java	A Delta Propagation quick start cache listener for simple uses.
DeltaReceiverListener.java	A Delta Propagation quick start receiver cache listener.

**Related Topics**[Delta Propagation](#) on page 317**Related Javadocs**

- [com.gemstone.gemfire.Delta](#)
- [com.gemstone.gemfire.cache.ClientCacheFactory](#)

**Function Execution Example**

With large data sets, moving function behavior to the application member hosting the data can provide better performance than moving data to the application that will run the function.

**Running the Example**

In this example of peer-to-peer function execution, one member sends a request for function execution to a peer member. FunctionExecutionPeer2 creates a region, populates the region and sends a function execution request to FunctionExecutionPeer1 while simultaneously executing the function on its own region. It collects the result from its own execution as well as from FunctionExecutionPeer1. The function executed is programmed in MultiGetFunction, as an implementation of GemFire's FunctionAdapter. The results from the function are handled by MyArrayListResultCollector, which is an implementation of GemFire's ResultCollector.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

1. In one session, start the first member:

```
$ java quickstart.FunctionExecutionPeer1
```

2. When the first member tells you to do so, start the second member in another session:

```
$ java quickstart.FunctionExecutionPeer2
```

**Example Source Files**

Program and cache configuration files for the function execution on JVMs 1 and 2. Both JVMs use the same cache XML configuration file.

**Table 37: Cache configuration files, located in SampleCode//quickstart/xml**

FunctionExecutionPeer.xml	Configures a region as a client region in a client/server cache.
---------------------------	--

**Table 38: Java program files, located in SampleCode//quickstart/quickstart**

FunctionExecutionPeer1.java	This is the peer to which FunctionExecutionPeer2 connects for function execution.
FunctionExecutionPeer2.java	Creates a region, populates the region, and sends a function execution request to FunctionExecutionPeer1 while simultaneously executing the function on its own region.
MultiGetFunction.java	Application Function to retrieve values for multiple keys in a region.
MyArrayListResultCollector.java	Gathers result from all members where the function is executed.

**Related Topics**

[Function Execution](#) on page 417

**Related Javadocs**

- `com.gemstone.gemfire.cache.execute`

**Secure Client Example**

You establish trust by verifying credentials when one process connects to another.

The security framework establishes trust between members, and authorizes client cache operations based on that trust. New members connect to the locator in a peer-to-peer topology, providing credentials to the locators. Clients connect to cache servers, providing credentials to the servers. One system connects to another in a multi-site system, using mutual authentication.

**Running the Example**

The example shows a very simple client/server configuration that uses security. The server starts on a port with security properties for client requests. The client does puts and gets on the server with valid PUT credentials. The client uses the valid LDAP username and password.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85. In addition, you will require an accessible LDAP server to perform authentication.

1. In one session, start the server:

```
$ java quickstart.SecurityServer ldap
ou=ldapserver,dc=mytown,dc=mycompany,dc=com
```

Replace the LDAP server location information with your LDAP server information.

2. In the other session start the client:

```
$ java quickstart.SecurityClient my_user my_password
```

Replace the username and password with a valid LDAP username and password.

3. Follow the instructions on the screens. When the client exits, press Enter to stop the server.

**Example Source Files**

Program and cache configuration files for the client and the server:

**Table 39: Cache configuration files, located in SampleCode/quickstart/xml**

SecurityServer.xml	Configures a cache to serve caching clients. The example region also is configured with a loader.
SecurityClient.xml	Configures a region as a client region in a Security cache.

**Table 40: Java program files, located in SampleCode/quickstart/quickstart**

SecurityServer.java	A server that starts on a port with security properties for client requests.
SecurityClient.java	Does put and get on the server with valid PUT credentials. This client uses the valid LDAP username and password.

**Related Topics**[Security](#) on page 497**Related Javadocs**

- [com.gemstone.gemfire.security](#)

**Multiple Secure Client Example**

In a client/server installation with security, you can create multiple, secure connections to your servers from a single client.

The most common use case is a GemFire client embedded in an application server that supports data requests from many users. Each user can be authorized to access a subset of data on the servers. For example, customer users may be allowed to see and update only their own orders and shipments. You create the multiple user connections from the single ClientCache instance in your client application. Each user provides its own credentials, and accesses the servers at its own individual authorization level.

**Running the Example**

This example shows a very simple client/server that uses multiuser authentication in the client. The server listens on a port for client requests and updates. The client does put and get on the server on behalf of multiple users with different credentials. This client uses the valid username and password.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

1. In one session, start the server:

```
$ java quickstart.MultiuserSecurityServer
```

2. In the other session start the client:

```
$ java quickstart.MultiuserSecurityClient
```

3. Follow the instructions on the screens. You can run the client multiple times trying different operations. Once you are done and the client has exited, press Enter to stop the server.

**Example Source Files**

Program and cache configuration source files for the client and the server:

**Table 41: Cache configuration files, located in SampleCode//quickstart/xml**

MultiuserSecurityServer.xml	Configures a cache to serve caching clients.
MultiuserSecurityClient.xml	Configures a region as a client region in a Security cache. The region's pool connects to the cacheserver.

**Table 42: Java program files, located in SampleCode/quickstart/quickstart**

MultiuserSecurityServer.java	This server starts on a port with security properties for client requests.
MultiuserSecurityClient.java	A client that does put and get on the server on behalf of multiple users with different credentials. This client uses the valid username and password.
MultiGetFunction.java	Application Function to retrieve values for multiple keys in a region.

**Related Topics**[Security](#) on page 497**Related Javadocs**

- [com.gemstone.gemfire.security](#) (Javadoc)

**Distributed Locking Example**

Use the distributed locking service to synchronize activities on shared resources.

With distributed locking, you program your applications to use an arbitrary named lock that only one application can own at a time. Use this to coordinate access to resources like shared files and non-locking data regions (those that do not have global scope).

**Running the Example**

This example uses two instances of the same application, run concurrently, to show how distributed locking can be used to obtain exclusive rights over a resource.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

For this example, the program runs through several iterations, trying to acquire a lock on a named lock service and updating a cached data entry when it gets the lock. All activities are reported to standard out.

Start the example at approximately the same time in your two terminal sessions, to create contention for the lock:

```
SESSION1
$ java quickstart.DistributedLocking
```

```
SESSION2
$ java quickstart.DistributedLocking
```

**Example Source Files**

Program and cache configuration files:

**Table 43: Cache configuration files, located in SampleCode/quickstart/xml**

DistributedLocking.xml	Configures a cache with a loader and listener.
------------------------	--

**Table 44: Java program files, located in SampleCode/quickstart/quickstart**

DistributedLocking.java	Demonstrates the use of distributed locking to manage concurrent access to resources.
-------------------------	---

### Related Javadocs

- [com.gemstone.gemfire.distributed.DistributedLockService](#)

## Benchmark Examples

The GemFire benchmarks measure the speed of data distribution between peers and between client and server.

You can run the test applications on the same machine or on separate machines. If you use separate machines, make sure both applications have access to a GemFire installation.

Example link	Description
<a href="#">Peer-to-Peer Benchmark Example</a> on page 108	Measures the time it takes to distribute an operation from one peer to another.
<a href="#">Client Server Benchmark Example</a> on page 109	Measures the time it takes to send an operation from a client to its server.

### Peer-to-Peer Benchmark Example

The P2P benchmark measures entry distribution time in a single system. A producer puts entries and measures the time it takes for a distribution acknowledgment from a consumer. For this example, the producer and consumer have configured their data region to require acknowledgment of data distribution. The consumer region is configured as a replica, so it automatically receives all of the producer's puts.

### Running the Example



**Note:** The tests run with a default number of samples, operations per sample, and payload. You can change these settings to better approximate your own systems. See [Benchmark Parameters](#) on page 110



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

1. In one session, start the consumer:

```
$ java quickstart.BenchmarkAckConsumer
```

2. After the consumer starts, in another session, start the producer:

```
$ java quickstart.BenchmarkAckProducer
```

### Example Source Files

Program and cache configuration files:

**Table 45: Cache configuration files, located in SampleCode//quickstart/xml**

BenchmarkAckProducer.xml	Configures a region for a distribution benchmark.
--------------------------	---

BenchmarkAckConsumer.xml	Configures a region with replication for a distribution benchmark.
--------------------------	--

**Table 46: Java program files, located in SampleCode//quickstart/quickstart**

BenchmarkAckProducer.java	Takes a timestamp, puts an entry value, waits for the ACK, and compares the current timestamp with the timestamp taken before the put.
BenchmarkAckConsumer.java	Creates the cache for the example.

## Client Server Benchmark Example

This example measures client-to-server entry distribution. The client puts data into its cache and measures the time it takes for the put to return. The measurements include the time required to send the data to the server.

The example is configured for client and server that are run on the same host machine. To run them on separate machines, under the client's quickstart directory, edit `xml/BenchmarkClient.xml`, locate the `<server>` element, and change "localhost" to your server's machine name or address. For example, if you run the server on a machine named freddy, the `<server>` element in the client's `xml/BenchmarkClient.xml` should look like this:

```
<server host="freddy" port="40404" />
```

### Running the Example



**Note:** The tests run with a default number of samples, operations per sample, and payload. You can change these settings to better approximate your own systems. See [Benchmark Parameters](#) on page 110.



**Note:** To run this example, you must have terminal sessions configured for the QuickStart examples, as described in [Setting Up Your Environment](#) on page 85.

1. In one session, start the server:

```
$ gfsh start server --name=server_bms  
--cache-xml-file=../xml/BenchmarkServer.xml
```

The `gfsh` command starts a cache server in the background, displays the status of the server and some environment information, and then exits to the command prompt.

2. Start the client session.

If you are running client and server on the same machine, you can start the client in the same session as the server. If you are running on separate machines, start a client session on the second machine. Use this command to start the client:

```
$ java quickstart.BenchmarkClient
```

The client executes some benchmarking operations and then displays the results. (The client can take several minutes to complete this task.)

3. After the client exits, stop the cache server in the server's session:

```
$ gfsh stop server --dir=server_bms
```

You can also use the deprecated `cacheserver` command to stop the cache server:

```
$ cacheserver stop
```

## Example Source Files

Program and cache configuration source files (the server is a GemFire cacheserver process and does not have an example source file):

**Table 47: Cache configuration files, located in SampleCode//quickstart/xml**

BenchmarkServer.xml	Configures a cache to serve caching clients at port 40404.
BenchmarkClient.xml	Configures a region as a client region in a client/server cache.

**Table 48: Java program files, located in SampleCode/quickstart/quickstart**

BenchmarkClient.java	Measures the time it takes for a client put to return in a hierarchical cache configuration.
----------------------	--

## Benchmark Parameters

In the benchmark tests, the producer (the client in the client/server example) runs a number of sampling iterations, each composed of a number of put() operations, with each put() operation writing a specific number of bytes. The settings used are reported to standard out at the start of each run. The default run uses 60 sampling iterations, with 5000 put operations per sample (thus the total 300,000 puts reported by the example), and each operation puts 10 kilobytes of data.

Run time is roughly determined by the number of samples multiplied by the number of operations per sample. The higher these combined properties are set, the longer the run time.

You can modify the benchmark to approximate the system you plan to implement. You can pass any combination of these properties to the producer:

Property	Description
benchmark.number-of-samples	Number of sampling iterations run by the producer. Default: 60
benchmark.operations-per-sample	Number of put operations performed for each sampling iteration. Default: 5000
benchmark.payload-bytes	Number of bytes of data cached by each put() operation. Default: 10240 (10 kilobytes).

For example, this client invocation sets the payload to 100 kilobytes, leaving the defaults for the other properties:

```
$ java -Dbenchmark.payload-bytes=102400 quickstart.BenchmarkClient
```

## Troubleshooting Checklist

This topic covers possible problems with running the examples and suggests corrective actions.

If your problems are not resolved here, please contact VMware technical support at <https://www.vmware.com/support/contacts/>.

## Connection Problems

If you get errors indicating you are unable to join the distributed system, you might have to change firewall settings or your distributed system connection information.

### Connection problems due to firewall restrictions

GemFire 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.

### Connection problems due to system member discovery conflict

By default, the examples use multicasting for system member discovery. You might run into connection problems because you are unable to use multicasting or because you need to use a different multicast port.

#### On Systems with Multicasting

If your system is configured for multicasting, connection problems might be due to contention between GemFire and non-GemFire applications for the same multicast port. GemFire is configured to use port 10334 by default. To change the port you use for the examples, edit the `gemfire.properties` file in the examples quickstart directory and change the setting for `mcast-port`. You might want to check with your system administrator to be sure you are switching to an unused port.

#### On Systems without Multicasting

If you are not able to use multicasting, you can use the GemFire locator process instead. To do this, start a GemFire locator to listen on a port of your choosing with the command. At a gfsh prompt, type:

```
gfsh>start locator --name=my_locator --port=yourPort
```

then modify the `gemfire.properties` file in the examples quickstart directory, setting the multicast port to zero (`mcast-port=0`) and the locators property to the host and port where you started the locator (`locators=locator_hostname[yourPort]`).

For example, you can start a locator (named locator1) with this command on a machine named "rosa":

```
gfsh>start locator --name=locator1 --port=10987
```

and modify the `gemfire.properties` file:

```
log-level=warning  
mcast-port=0  
locators=rosa[10987]
```

### Exception in thread "main" java.lang.NoClassDefFoundError

If you have problems finding the classes used in the examples, verify that you have completed all environment configuration steps in [Installing vFabric GemFire](#) on page 27 and [Setting Up the Product Examples](#) on page 61, so Java can find the product jar files and the quick start example classes.

### Exception . . . CacheXML file/resource . . . does not exist

If you have problems finding the xml files used in the examples, verify that you have completed all environment configuration steps in [Installing vFabric GemFire](#) on page 27 and [Setting Up the Product Examples](#) on page 61 and that you are in the example installation quickstart directory, the parent directory to the xml directory where

the cache configuration files are stored. If you do not have your environment configured properly or you are not in the quickstart directory, you may see errors like this:

```
Exception in thread "main" com.gemstone.gemfire.cache.CacheXmlException:  
Declarative Cache XML file/resource "xml\BenchmarkHierarchicalServer.xml"  
does not exist
```

### **Exception . . . Cannot create region**

If program startup fails with an exception similar to the one listed below, a conflict may exist with an example that is already running.

```
Exception in thread "main" java.lang.IllegalStateException: Cannot  
create region /root/exampleRegion with DISTRIBUTED_ACK scope because  
another cache has same region with DISTRIBUTED_NO_ACK scope at  
com.gemstone.gemfire.internal.cache.CreateRegionProcessor$  
CreateRegionMessage.process(CreateRegionProcessor.java:163)
```

This conflict can occur if two examples are run at the same time with the same distributed system settings and different example region configuration. Distributed regions must satisfy consistency requirements across the system. Thus if you try to create a region that is already defined by another process in the system and your new region configuration conflicts with the existing one, the creation fails.

Check that you have not left another example running in one of your sessions. If you suspect that you are running your examples in the same distributed system as another person, change the mcast-port setting in the example installation quickstart directory's `gemfire.properties` file.

# Part 3

---

## Getting Started with Spring Data GemFire

*Getting Started with Spring Data GemFire* describes how experienced Spring developers can use Spring Data GemFire to integrate vFabric GemFire components into their applications.

Overviews of GemFire functionality, recommended Spring techniques for integration, and code examples are provided, as well as links to more detailed information.



**Note:** Some common configurations that developers use in Spring Data GemFire applications are described here. However, for a complete discussion of using Spring Data GemFire in your applications, see the [Spring Data Gemfire Reference Guide](#). This guide, *vFabric GemFire User's Guide*, provides complete documentation for configuring, managing, and using vFabric GemFire. For a high-level discussion of vFabric GemFire, see [About vFabric GemFire](#) on page 21.

### Topics:

- [About vFabric GemFire](#)
- [About Spring Data GemFire](#)
- [Configuring GemFire Components with the Spring Framework](#)



# Chapter 16

---

## About vFabric GemFire

vFabric GemFire is an in-memory data-management solution designed to eliminate disk access as the limiting factor for data-intensive applications.

GemFire manages data in memory and the data is distributed across a cluster of nodes to provide improved performance, scalability, and fault tolerance. Where required, GemFire can synchronize the data with external disk-based databases.

### Main Concepts

This section describes high-level concepts that are key to understanding vFabric GemFire.

*Caches* are an abstraction that describe a node in a GemFire distributed system. Application architects can arrange these nodes in several deployment architectures, including: peer to peer, client/server, and WAN gateway topologies.

Within each cache, you define data *regions*. Data regions are analogous to tables in a relational database and manage data in a distributed fashion as name/value pairs. A *replicated* region stores identical copies of the data on each cache member of a distributed system. A *partitioned* region spreads the data among cache members. After the system is configured, client applications can access the distributed data in regions without knowledge of the underlying system architecture. You can define listeners to create notifications about when data has changed, and you can define expiration criteria to delete obsolete data in a region.

GemFire uses *continuous querying* to enable event-driven architectures. GemFire ties events and data together so that when an event is processed, the data required to process the event is available without additional queries to a disk-based database. Clients can subscribe to change notifications so that they can execute tasks when a specific piece of data changes.

By default, GemFire clients and servers use port 40404 and multicast to discover each other. For large, production systems, GemFire provides *locators*. Locators provide both discovery and load balancing services. You configure clients with a list of locator services and the locators maintain a dynamic list of member servers.

For more information, see [Basic Configuration and Programming](#) on page 131.



# Chapter 17

---

## About Spring Data GemFire

The Spring Data GemFire project allows developers to use the Spring Framework and vFabric GemFire to create and configure a distributed system for data-centric applications.

The framework also allows developers to separate the modules that access and manipulate this data from those that define and manage the distributed system.

Once the system is configured and deployed, application code can use standard data access techniques to write the business logic code that uses the data.

### Overview of Application Development with Spring Data GemFire

In vFabric GemFire, a common architecture that separates the business logic from the system configuration is the *client/server* architecture. In this type of system, client application processes are cache clients that produce and consume data while server processes manage the distribution of data among the nodes of the system.

This type of architecture allows for a scalable, highly-available, low latency system where a farm of cache servers provide services to many clients.

For example, consider the `basic/replicated-cs` code example (See [Spring Data GemFire Code Examples on page 119](#).) In the this example, The `Server.java` class, and the `server/cache-config.xml` file define a cache, its associated properties, a cache-server, a replicated region called **Customer**, and an associated cache listener. When this module is deployed, it defines a server node that can service data requests for the **Customer** data region.



**Note:** The `cache-config.xml` file discussed in this section is a Spring Framework application context XML file that is using the Spring Data GemFire namespace. See [Configuring GemFire Components with the Spring Framework](#) on page 121.

The `server/cache-config.xml` file contains the following configurations:

```
<util:properties id="gemfire-props">
    <prop key="log-level">warning</prop>
</util:properties>

<gfe:cache properties-ref="gemfire-props" />

<gfe:cache-server/>

<gfe:replicated-region id="Customer">
    <gfe:cache-listener>
        <bean
            class="org.springframework.data.gemfire.examples.util.LoggingCacheListener" />
```

```
</gfe:cache-listener>
</gfe:replicated-region>
```

The Server.java class only needs to reference the **server/cache-config.xml** file:

```
public class Server {
    public static void main(String args[]) throws IOException {
        new ClassPathXmlApplicationContext("server/cache-config.xml");
    }
}
```

The following configuration for the client creates a client cache, client region, and a connection pool to get data from the **Customer** region that was defined for the server.

```
<util:properties id="gemfire-props">
    <prop key="log-level">warning</prop>
</util:properties>

<gfe:client-cache properties-ref="gemfire-props" />

<gfe:client-region id="Customer" data-policy="empty" />

<gfe:pool>
    <gfe:server host="localhost" port="40404" />
</gfe:pool>
```

The Client.java class uses the client cache to update data in the **Customer** region.

```
public class Client {

    @SuppressWarnings("unchecked")
    public static void main(String args[]) throws IOException {

        ApplicationContext context = new
ClassPathXmlApplicationContext("client/cache-config.xml");
        Region<Long, Customer> region = context.getBean(Region.class);

        Customer dave = new Customer(1L,new
EmailAddress("dave@matthews.com"),"Dave","Matthews");
        Customer alicia = new Customer(2L,new
EmailAddress("alicia@keys.com"),"Alicia","Keys");
        region.put(dave.getId(),dave);
        region.put(alicia.getId(),alicia);

    }
}
```

You can enable subscriptions for client regions to synchronize data from a region defined in a cache server. The client can register interest in certain keys to receive notifications when data changes. To balance usage and improve performance, a client region uses connection pools to connect to a cache server.

For complete documentation of Spring Data GemFire, see the [Spring Data GemFire Reference Guide](#). This guide, *vFabric GemFire User's Guide*, provides complete documentation of vFabric GemFire.

## Recommended Development Model

The Spring development environment provides several programming models that allow developers to create and configure Java applications that use GemFire caches and regions for high-performance data access. These models include using Spring dependency injection with the standard GemFire `cache.xml` file to configure GemFire; using the `GemFireTemplate` (similar to the Spring Data `JDBCTemplate`); and using the Spring XML Namespace to configure GemFire data access.

VMware recommends that developers use the Spring XML Namespace to configure their GemFire components and this chapter provides an overview of that model. For information on other programming models, see the [Spring Data GemFire Reference Guide](#).

Using the Spring XML namespace provides the following advantages:

- Developers can architect their applications so that data access and business logic are separated from configuration and operation code.
- Developers can use Spring property management features and Spring Expression Language (SpEL) to configure their applications.
- Developers can use Spring Framework features such as transaction management and modular configuration in their applications.
- The Spring Tool Suite (STS) provides support for creating and editing Spring XML namespaces and Java code, including code completion, pop-up annotations, and validation. You can also use other IDEs.
- Developers can replace the standard GemFire `cache.xml`, which provides limited configuration options, with the Spring XML application context configuration. This also eliminates the requirement for callbacks that implement the GemFire `Declarable` interface, which is not required when using Spring.

## Spring Data GemFire Code Examples

A set of Java code examples demonstrate coding of Spring Data GemFire applications. The examples include Java source files, Spring application context XML configuration files, and the gradle utility. The gradle utility downloads the `gemfire.jar` file, (the GemFire API) the Spring Framework, and Spring Data Gemfire.

The examples are ready to run using the provided instructions. (See the `README.md` files in each example directory for build and run instructions.) Java Development Kit (JDK) 1.6 or later is required to build and run the examples.

The advanced examples (`advanced/gateway` and `advanced/locator-failover`) require a full installation of GemFire. You may use a trial license to run the examples.

The examples download also includes an instance of the `gradle` build tool that you can use to run the examples. You can also run a `gradle` task that creates Eclipse project files that you can import into the STS or Eclipse IDEs. (Gradle is a build automation tool similar to Ant or Maven and is included with the examples.)

The examples are also available from GitHub at the following URL:

<http://github.com/SpringSource/spring-gemfire-examples.git>

You can also use the following `git` command (a `git` client is required) to download the examples:

```
git clone git://github.com/SpringSource/spring-gemfire-examples.git
```

The "Additional Resources" sections within each configuration topic list examples directories where you can find the full code.

## Additional Resources

*Getting Started with Spring Data GemFire* does not provide a complete reference on Spring Data GemFire or vFabric GemFire. Additional resources are listed here.

### Spring Data GemFire Resources

- [Spring Data GemFire Home Page](#)
- [Spring Data GemFire Reference Guide](#)
- [Spring Data GemFire API \(Javadocs\)](#)
- [Spring Data GemFire Forum](#)
- [Spring Data GemFire Examples](#)

- [Spring Data: Modern Data Access for Enterprise Java](#) (published by O'Reilly Media). See chapter 14, "GemFire - A Distributed Data Grid".

### vFabric GemFire Resources

- [Getting Started with vFabric GemFire](#) on page 15 (part of this guide, *vFabric GemFire Users Guide*)
- [vFabric GemFire Resources](#) page (technical white papers, data sheet, best practices, Webcasts and more)
- [vFabric GemFire Community page](#)
- [GemFire Java API \(Javadocs\)](#)

# Chapter 18

---

## Configuring GemFire Components with the Spring Framework

You can configure vFabric GemFire through a Spring application-context XML file that uses the Spring Data GemFire XML namespace.

### System Requirements

- Make sure your system meets the requirements to develop and deploy applications with Spring Data GemFire.
- vFabric GemFire, version 6.6 or later.
  - Java Development Kit (JDK) version 1.6 or later
  - Spring Data GemFire, version 1.2.1
  - Spring Framework, version 3.1
  - (optional) SpringSource Tool Suite (STS). The STS is an Eclipse-based IDE for creating and editing Spring modules. You can also use other IDEs, such as IntelliJ or standard Eclipse.



**Note:** In addition to the preceding requirements, make sure your system meets the requirements to run vFabric GemFire. See [Supported Configurations and System Requirements](#) on page 1.

### Downloading vFabric GemFire and Spring Data GemFire Components

This topic describes how to obtain the components for using Spring Data GemFire.

You can obtain the necessary components to develop applications from the following locations, or you can obtain them from a public repository by using the Maven or Gradle build tools to specify the repository and dependencies.

- [Spring Framework download](#)
- [vFabric GemFire download](#)
- [Spring Data GemFire download](#)
- [SpringSource Tool Kit \(STS\) download](#) (optional)

Follow the instructions included with each download to install the components on your computer.



**Note:** If you download the Spring Data GemFire examples from GitHub, the download includes gradle scripts that download the `gemfire.jar` file, and the Spring Framework and Spring Data Gemfire jar files. (See [Spring Data GemFire Code Examples](#) on page 119.)

## Obtaining Spring Data Gemfire and GemFire from a Public Repository

You can use the Maven or Gradle utilities to download the Spring Framework, the `gemfire.jar` file, and Spring Data GemFire. For example, in Maven, in the `pom.xml` file, add the following repository to download the `gemfire.jar` file and the dependency on Spring Data GemFire:

```
<repositories>
  <repository>
    <id>spring-plugins-repository</id>
    <name>Spring Artifactory Plugins(Dependencies) Repository</name>
    <url>http://repo.springsource.org/plugins-release</url>
  </repository>
</repositories>

<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-gemfire</artifactId>
  <version>1.2.2.RELEASE</version>
</dependency>
```



**Note:** The `gemfire.jar` file that is available from the repositories or downloaded by the gradle scripts includes only the GemFire Java API and is not a full installation of GemFire. See [Installing vFabric GemFire](#) on page 27 for instructions on how to obtain a full installation.

## Configuring Caches

To create a cache, you include the cache declaration in your Spring application context XML configuration file. Because this declaration contains no properties, the default configuration is used.

```
<gfe:cache />
```

The Spring application context registers a `CacheFactoryBean` that creates a Spring bean named `gemfireCache`. This Spring bean references a GemFire Cache instance. You can add an `id` property to override the default name.

You can also specify GemFire properties that configure the cache using standard Spring features. For example, the following declaration in a Spring application context XML file uses the `Spring util` namespace to reference an external properties file:

```
...
  <gfe:cache properties-ref="props" />

  <util:properties
    id="props"
    location="file:/vfabric/gemfire/gemfire.properties" />
...
```

There are additional properties you can use to configure memory usage, serialization, timeouts, and more. For a description of these properties and other configuration options, see [Spring Data Gemfire Schema](#).

### Additional Resources:

- [Spring Data GemFire Reference Guide: Configuring the GemFire Cache](#)
- [Cache Management](#) on page 137
- See the following Spring Data GemFire examples:
  - `basic/replicated`
  - `quickstart/spring-cache`

## Configuring Regions

After you configure a cache, you can configure a region. There are four types of regions you can create.

- **replicated-region**

In a replicated region, each member holds all data defined for the region

- **partitioned-region**

In a partitioned region, portions of the data are held in each member. You configure a redundant copy of the data for failover scenarios.

- **local-region**

A local region does not share data with other member caches.

- **client-region**

Client regions connect to a cache server using a connection pool. See [Configuring a Client/Server Architecture](#) on page 125.

For more information on the types of regions you can create, see [Region Types](#) on page 206 in the *vFabric GemFire User's Guide*.

For example, the following Spring application context XML configuration (from the `basic/replicated` example) creates a cache that uses the default `id` (`gemfireCache`) to create a replicated region named `Customer`. You can supply a valid bean name for the cache by specifying the name with the `id` attribute and then referencing the `id` with the `cache-ref` attribute of the region declaration.

```
...
<util:properties id="gemfire-props">
    <prop key="log-level">warning</prop>
</util:properties>

<gfe:cache properties-ref="gemfire-props" />

<gfe:replicated-region id="Customer">
    <gfe:cache-listener>
        <bean
            class="org.springframework.data.gemfire.examples.util.LoggingCacheListener" />
    </gfe:cache-listener>
</gfe:replicated-region>

...

```

The following Java class populates a `Customer` object from the `Customer` region that is defined in the above XML declarations:

```
public class Consumer {
    private static Log log = LogFactory.getLog(Consumer.class);

    @SuppressWarnings("unchecked")
    public static void main(String args[]) throws IOException {

        ApplicationContext context =
            new ClassPathXmlApplicationContext("consumer/cache-config.xml");

        Region<Long, Customer> region = context.getBean(Region.class);

        Customer dave = region.get(1L);
    }
}
```

```

        if (dave != null) {
            log.debug("retrieved " +
                dave.getFirstname() + " " +
                dave.getLastname());
        }

        Customer alicia = region.get(2L);
        if (alicia != null) {
            log.debug("retrieved " +
                alicia.getFirstname() + " " +
                alicia.getLastname());
            alicia.add(new Address("Keys Street", "Alicia", "UK"));
            region.put(alicia.getId(), alicia);
        }
    }

}

```

### Additional Resources

- [Spring Data GemFire Reference Guide: Configuring a GemFire Region](#)
- [Region Data Storage and Distribution](#) on page 205
- Spring Data GemFire examples:
  - basic/replicated
  - basic/partitioned

## Configuring Cache Listeners

You can register *Cache Listeners* for a region to handle data events such as when data is created, updated, or deleted. A Cache Listener is a bean that implements the `com.gemstone.gemfire.cache.CacheListener` interface.

The following configuration declares a cache listener for the `Customer` region:

```

<gfe:replicated-region id="Customer">
    <gfe:cache-listener>
        <bean
class="org.springframework.data.gemfire.examples.util.LoggingCacheListener"/>

    </gfe:cache-listener>
</gfe:replicated-region>

```

The following Java class logs changes to region data. The class extends the `GemFireCacheListenerAdapter` class, which is an implementation of the `CacheListener` interface.

```

public class LoggingCacheListener extends CacheListenerAdapter {
    private static Log log = LogFactory.getLog(LoggingCacheListener.class);

    @Override
    public void afterCreate(EntryEvent event) {
        final String regionName = event.getRegion().getName();
        final Object key = event.getKey();
        final Object newValue = event.getNewValue();
        log.info("In region [" + regionName + "] created key [" + key
            + "] value [" + newValue + "]");
    }

    @Override

```

```

public void afterDestroy(EntryEvent event) {
    final String regionName = event.getRegion().getName();
    final Object key = event.getKey();
    log.info("In region [" + regionName + "] destroyed key [" + key
        + "]");
}

@Override
public void afterUpdate(EntryEvent event) {
    final String regionName = event.getRegion().getName();
    final Object key = event.getKey();
    final Object newValue = event.getNewValue();
    final Object oldValue = event.getOldValue();
    log.info("In region [" + regionName + "] updated key [" + key
        + "] [oldValue [" + oldValue + "] new value [" + newValue + "]]");
}
}

```

## Additional Resources

- [Spring Data GemFire Reference Guide: Cache Listeners](#)
- [Spring Data GemFire example:](#)

`spring-gemfire-examples-common/common/src/main/java/org/springframework/
 data/gemfire/examples/util/LoggingCacheListener.java`

(This Java class is used by several of the examples and is therefore located in the `common` directory.)

## Configuring a Client/Server Architecture

To use the GemFire client/server architecture, you configure both *cache servers* and *client caches*.

Cache servers operate in a peer-to-peer relationship with other cache servers. Multiple client processes can connect to cache servers to retrieve and update data. Clients connect using connection pools.

The following Spring application context XML configuration (from the `basic/replicated-cs` example) declares a cache server:

```
<gfe:cache-server/>
```

The following Java class (from the `basic/replicated-cs` example) references the Spring application context XML configuration file to create the server:

```

public class Server {
    public static void main(String args[]) throws IOException {
        ...
        new ClassPathXmlApplicationContext("server/cache-config.xml");
        ...
    }
}
```

You configure a GemFire *client cache* to access data from a cache server. In this type of architecture, many clients can connect to a smaller number of cache servers. The clients can use connection pools to connect to the servers.

The following Spring application context XML configuration (from the `basic/replicated-cs` example) creates a client cache:

```
<gfe:client-cache />
```

The following Java code (from the `basic/replicated-cs` example) uses a client cache to update data:

```
public class Client {

    @SuppressWarnings("unchecked")
    public static void main(String args[]) throws IOException {

        ApplicationContext context =
            new ClassPathXmlApplicationContext("client/cache-config.xml");
        Region<Long, Customer> region = context.getBean(Region.class);

        Customer dave =
            new Customer(1L,
                         new EmailAddress("dave@matthews.com"),
                         "Dave",
                         "Matthews");
        Customer alicia =
            new Customer(2L,
                         new EmailAddress("alicia@keys.com"),
                         "Alicia",
                         "Keys");
        region.put(dave.getId(), dave);
        region.put(alicia.getId(), alicia);

    }
}
```

### Additional Resources

- [Spring Data GemFire Reference Guide: Configuring a GemFire Cache Server](#)
- [Spring Data GemFire Reference: Configuring a GemFire Client Cache](#)
- [Client/Server Configuration](#) on page 177
- Spring Data GemFire example:

`basic/replicated-cs`

## Configuring Continuous Querying

Continuous querying allows you to create a query and then receive notifications when new data is added that matches the query. Continuous querying requires a client/server topology and the pool must enable the `subscription` property.

The following client Spring application context XML configuration (from the `quickstart/cq` example) declares a pool and a continuous query listener container, and enables continuous query for a POJO (the `cqListener` bean).

```
<gfe:client-cache pool-name="client-pool" properties-ref="gemfire-props"/>

<gfe:pool id="client-pool" subscription-enabled="true">
    <gfe:server host="localhost" port="40404"/>
</gfe:pool>

<gfe:cq-listener-container>
    <gfe:listener
        ref="cqListener"
```

```

        query="select * from /Order where customerId = 2" />
</gfe:cq-listener-container>

<bean id="cqListener"
      class="org.springframework.data.gemfire.examples.CQListener" />

```

The following Java class (from the quickstart/cq example) creates a continuous query listener from the POJO defined in the XML configuration.

```

public class CQListener {

    private static Log log = LogFactory.getLog(CQListener.class);

    public void handleEvent(CqEvent event) {
        log.info("Received a CQ event " + event);
    }
}

```

## Additional Resources

- [Spring Data GemFire Reference Guide: Continuous Query Container](#)
- [Continuous Querying](#) on page 375
- Spring Data GemFire Example:  
quickstart/cq

## Configuring Persistence

You can persist the data that your application stores in a region. Persistence allows you to restore data after a failure or normal shutdown of GemFire.

To configure persistence, you set the `persistent` attribute of your region configuration to `true`. You also set a `disk-store` attribute that defines where the data is persisted.

For example, the following Spring application context XML configuration (from the basic/persistence example) enables persistence for a partitioned region that uses a disk store:

```

<gfe:replicated-region id="Order" disk-store-ref="orderDiskStore" >
    <gfe:cache-listener ref="cacheListener"/>
    <gfe:eviction
        threshold="300"
        type="ENTRY_COUNT"
        action="OVERFLOW_TO_DISK" />
</gfe:replicated-region>
...
<gfe:disk-store id="orderDiskStore">
    <gfe:disk-dir location="orders" />
</gfe:disk-store>

```

## Additional Resources

- [Spring Data GemFire Reference: Data Persistence](#)
- [Spring Data GemFire Reference: Configuring a Disk Store](#)
- [Persistence and Overflow](#) on page 245
- Spring Data GemFire example:  
basic/persistence

## Configuring Transactions

The Spring framework provides a transaction programming model that can work across multiple APIs. GemFire provides a per-cache transaction manager that you can use to execute region data operations atomically.

For example, the following Spring application context XML configuration (from the quickstart/transaction example) declares a cache, region, and transaction manager using default names. (gemfireCache) for the cache and gemfireTransactionManager for the transaction manager):

```
<gfe:cache properties-ref="gemfire-props"/>
<gfe:replicated-region id="Customer"/>
<gfe:transaction-manager/>
```

The following Java class (from the quickstart/transaction example) references the transaction manager:

```
public class CustomerService {

    @Autowired CustomerRepository customerRepository;

    /**
     * @param customer
     * @param fail - set to true to simulate some downstream processing that
     *   throws an exception which should trigger a rollback
     */
    @Transactional("gemfireTransactionManager")
    public void updateCustomer(Customer customer, boolean fail) {
        customerRepository.save(customer);
        if (fail) {
            throw new RuntimeException(
                "Updated failed - should trigger a rollback.");
        }
    }

    public Customer getCustomer(long id) {
        return customerRepository.findOne(id);
    }
}
```

### Additional Resources

- [Spring Data GemFire Reference: Transaction Management](#)
- [Transactions](#) on page 383
- Spring Data GemFire example:  
quickstart/transaction

## Configuring Functions

You can register GemFire functions for remote execution. You declare GemFire functions as Spring beans, which must implement the com.gemstone.gemfire.cache.execute.Function interface or extend the com.gemstone.gemfire.cache.execute.FunctionAdapter class.

For example, the following Spring application context XML configuration (from the basic/function example) declares a function:

```
<gfe:function-service>
    <gfe:function ref="calculateTotalSalesForProduct" />
</gfe:function-service>
```

The following Java class (from the basic/function example) extends the `FunctionAdapter` class to define the `CalculateTotalSalesForProduct` function:

```
public class CalculateTotalSalesForProduct extends FunctionAdapter {
    private static Log log =
    LogFactory.getLog(CalculateTotalSalesForProduct.class);

    @Resource(name="productTemplate")
    private GemfireTemplate productTemplate;
    @Resource(name="Order")
    private Region<Long,Order> orderRegion;

    /* (non-Javadoc)
     * @see com.gemstone.gemfire.cache.execute.FunctionAdapter#
     *      execute(com.gemstone.gemfire.cache.execute.FunctionContext)
     */
    @Override
    public void execute(ExecutionContext functionContext) {
        ResultSender<BigDecimal> resultSender = functionContext.getResultSender();

        String productName = (String)functionContext.getArguments();
        log.debug("searching for product name '" + productName + "'");

        SelectResults<Product> results =
            productTemplate.query("name = '" + productName + "'");

        if (results.isEmpty()) {
            log.warn("cannot find product '" + productName + "'");
            resultSender.lastResult(new BigDecimal(0.0));
            return;
        }

        Product product = results.asList().get(0);
        long productId = product.getId();

        BigDecimal total = new BigDecimal(0.0);

        for (Order order: orderRegion.values()) {
            for (LineItem lineItem: order.getLineItems()) {
                if (lineItem.getProductId() == productId) {
                    total = total.add(lineItem.getTotal());
                }
            }
        }

        resultSender.lastResult(total.setScale(2,BigDecimal.ROUND_CEILING));
    }

    /* (non-Javadoc)
     * @see com.gemstone.gemfire.cache.execute.FunctionAdapter#getId()
     */
    @Override
    public String getId() {
        return getClass().getSimpleName();
    }
}
```

```
}
```

### Additional Resources

- [Function Execution](#) on page 417
- <http://static.springsource.org/spring-data/gemfire/docs/current/reference/html/bootstrap.html#bootstrap:function>
- Spring Data GemFire example:  
[basic/function](#)

## Additional Spring Data GemFire Examples

Other Spring Data GemFire examples demonstrate additional features, which are not discussed in this chapter.

**Table 49: Additional GemFire Features**

Example	Feature Description	Additional Resources
advanced/gateway	WAN gateways allow you to synchronize vFabric GemFire data across geographically distributed areas.	<ul style="list-style-type: none"> <li>• <a href="#">Spring Data GemFire Reference guide: Configuring WAN Gateways</a></li> <li>• <a href="#">Multi-site (WAN) Configuration</a> on page 189</li> </ul>
advanced/locator-failover	If a GemFire locator service fails, you can configure fail over to another member.	<ul style="list-style-type: none"> <li>• <a href="#">vFabric GemFire User's Guide: vFabric GemFire Locator Process</a></li> <li>• <a href="#">vFabric GemFire Locator Process</a> on page 737</li> </ul>
quickstart/gemfire-template	Use the Spring Data GemFireTemplate to access region data. The Gemfire template wraps a native GemFire Region and translates GemFire checked exceptions and native runtime exceptions into a Spring DataAccessExceptions RuntimeException hierarchy.	<ul style="list-style-type: none"> <li>• <a href="#">Spring Data GemFire Reference Guide: GemFire Template</a></li> </ul>
quickstart/repository	Use the Spring Data repository abstraction to easily persist entities into GemFire and execute queries.	<ul style="list-style-type: none"> <li>• <a href="#">Spring Data GemFire Reference Guide: GemFire Repositories</a></li> </ul>
quickstart/spring-cache	Use the Spring Framework @Cacheable annotation with GemFire.	<ul style="list-style-type: none"> <li>• <a href="#">Spring Data GemFire Reference Guide: Support for Spring Cache Abstraction</a></li> </ul>

# Part 4

---

## Basic Configuration and Programming

*Basic Configuration and Programming* describes how to configure distributed system and cache properties for your vFabric GemFire installation. For your applications, it provides guidance for writing code to manage your cache and distributed system connection, data regions, and data entries, including custom classes.



**Note:** Although this section primarily discusses using the GemFire `cache.xml` file to configure GemFire and using the GemFire API to write applications, Java developers can also use the Spring Data GemFire API to configure a distributed GemFire system and write applications that access and use data stored in GemFire. Spring Data GemFire provides many advantages for creating such applications with GemFire.

For more information on the advantages of using Spring with GemFire, see [Getting Started with Spring Data GemFire](#) on page 113.

For a complete discussion of Spring Data GemFire, see [Spring Data GemFire Reference Guide](#)

### Topics:

- [Distributed System and Cache Configuration](#)
- [Cache Management](#)
- [Data Regions](#)
- [Data Entries](#)



# Chapter 19

---

## Distributed System and Cache Configuration

To work with your vFabric GemFire applications, you use a combination of configuration files and application code

### Distributed System Members

Distributed system members are programs that connect to a vFabric GemFire distributed system. You configure members to belong to a single distributed system, and you can optionally configure them to be clients or servers to members in other distributed systems, and to communicate with other distributed systems.

Distributed system members (or simply "members") connect to the GemFire system when they create the GemFire data cache. The members' distributed system is configured through GemFire properties. See [gemfire.properties and gfsecurity.properties \(GemFire Properties\)](#) on page 851. GemFire properties specify all necessary information for system member startup, initialization, and communication.



**Note:** You cannot change a member's properties while the member is connected to the distributed system.

Use the properties to define:

- Non-default licensing
- How to find and communicate with other system members
- How to perform logging and statistics activities
- Licensing information for the system
- What `cache.xml` file to use for cache and data region initialization
- Other options, including event conflation, how to handle network loss, and security settings

### Distributed System Membership and System Topologies

Every GemFire process is a member of a distributed system, even if the distributed system is defined as standalone, with just one member. You can run an individual distributed system in isolation or you can combine systems for vertical and horizontal scaling. See [Topology and Communication General Concepts](#) on page 163.

- **Peer-to-Peer Distributed Systems.** Members that define the same member discovery properties belong to the same distributed system and are peers to one another.
- **Client/Server Installations.** The client/server topology uses relationships that you configure between members of multiple distributed systems. You configure some or all of the peers in one distributed system to act as cache servers to clients connecting from outside the system. Each server can host many client processes, managing cache access for all in an efficient, vertically hierarchical cache configuration. You configure the client applications to connect to the servers, using a client cache configuration. Clients run as members of standalone GemFire distributed systems, with no peers, so all data updates and requests go to the servers.

- **Multi-site Installations.** The multi-site topology uses relationships that you configure between members of multiple distributed systems. Through this configuration, you loosely couple two or more distributed systems for automated data distribution. This is usually done for sites at geographically separate locations. You configure a subset of peers in each distributed system site with gateway senders and/or gateway receivers to manage events that are distributed between the sites.

## Options for Configuring the Distributed System

GemFire provides a default distributed system configuration, for out-of-the-box systems. To use non-default configurations and to fine-tune your member communication, you will customize your system configuration. You can use a mix of various options for customizing your distributed system configuration.

GemFire properties are used to join a distributed system and configure system member behavior. Configure your GemFire properties through the `gemfire.properties` file, the Java API, or command-line input. Generally, you 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 from keyboard input.



**Note:** Check with your GemFire system administrator before changing properties through the API, including the `gemfire.properties` and `license-*` settings. The system administrator may need to set properties at the command line or in configuration files. Any change made through the API overrides those other settings.



**Note:** The product `defaultConfigs` directory has a sample `gemfire.properties` file with all default settings.

Set distributed system properties by any combination of the following. The system looks for the settings in the order listed:

1. `java.lang.System` property setting. Usually set at the command line. 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`.

- In 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 cache server, pass the properties on the gfsh command line as command-line options. Example:

```
gfsh>start server --name=server_name --mcast-port=10338  
--cache-xml-file=serverConfig/cache.xml
```

See [vFabric GemFire Servers](#) on page 741 for more information on running cache servers.

3. Entry in a gemfire.properties file. See [Deploying vFabric GemFire Configuration Files](#) on page 631. Example:

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

4. Default value. The default property values are listed in the online Java documentation for com.gemstone.gemfire.distributed.DistributedSystem.

## Options for Configuring the Cache and Data Regions

To populate your vFabric GemFire cache and fine-tune its storage and distribution behavior, you need to define cached data regions and provide custom configuration for the cache and regions.

Cache configuration properties define:

- Cache-wide settings such as disk stores, communication timeouts, gateway senders, gateway receivers, and settings designating the member as a server
- Cache data regions

Configure the cache and its data regions through one or both of these methods:

- Through declarations in the XML file named in the cache-xml-file gemfire.properties setting. This file is generally referred to as the cache.xml file, but it can have any name. See [cache.xml](#) on page 861
- Through application calls to the com.gemstone.gemfire.cache.CacheFactory, com.gemstone.gemfire.cache.Cache and com.gemstone.gemfire.cache.Region APIs.

## Local and Remote Membership and Caching

For many vFabric GemFire discussions, you need to understand the difference between local and remote membership and caching.

Discussions of GemFire membership and caching activities often differentiate between local and remote. Local caching always refers to the central member under discussion, if there is one such obvious member, and remote refers to other members. If there is no clear, single local member, the discussion assigns names to the members to differentiate. Operations, data, configuration, and so forth that are "local to member Q" are running or resident inside the member Q process. Operations, data, configuration, and so on, that are "remote to member Q" are running or resident inside some other member.

The local cache is the cache belonging to the local member. All other caches are remote, whether in other members of the same distributed system or in different distributed systems.

In the context of a single distributed system, unless otherwise specified, remote refers to other members of the same distributed system. In client/server and multi-site installations, remote generally refers to members in other distributed systems. For example, all servers are remote to the clients that connect to them. Each client runs standalone, with connections only to the server tier, so all servers and their other clients are remote to the individual client. All gateway receivers are remote to the gateway senders that connect to them from other distributed systems, and to those gateway senders' peers.



# Chapter 20

---

## Cache Management

The GemFire cache is the entry point to GemFire caching management. GemFire provides different APIs and XML configuration models to support the behaviors of different members.

### Introduction to Cache Management

The cache provides in-memory storage and management for your data.

You organize your data in the cache into *data regions*, each with its own configurable behavior. You store your data into your regions in key/value pairs called *data entries*. The cache also provides features like transactions, data querying, disk storage management, and logging. See the Javadocs for `com.gemstone.gemfire.cache.Cache`.

You generally configure caches using a combination of XML declarations and API calls. GemFire loads and processes your XML declarations when you first create the cache.

GemFire has one cache type for managing server and peer caches and one for one for managing client caches. The `cacheserver` process automatically creates its server cache at startup. In your application process, the cache creation returns an instance of the server/peer or client cache. From that point on, you manage the cache through API calls in your application.

### The Caching APIs

GemFire's caching APIs provide specialized behavior for different system member types and security settings.

- **`com.gemstone.gemfire.cache.RegionService`**. Generally, you use the `RegionService` functionality through instances of `Cache` and `ClientCache`. You only specifically use instances of `RegionService` for limited-access users in secure client applications that service many users. The `RegionService` API provides access to existing cache data regions and to the standard query service for the cache. For client caches, queries are sent to the server tier. For server and peer caches, queries are run in the current cache and any available peers. `RegionService` is implemented by `GemFireCache`.
- **`com.gemstone.gemfire.cache.GemFireCache`**. You do not specifically use instances of `GemFireCache`, but you use `GemFireCache` functionality in your instances of `Cache` and `ClientCache`. `GemFireCache` extends `RegionService` and adds general caching features like region attributes, disk stores for region persistence and overflow, and access to the underlying distributed system. `GemFireCache` is implemented by `Cache` and `ClientCache`.
- **`com.gemstone.gemfire.cache.Cache`**. Use the `Cache` interface to manage server and peer caches. You have one `Cache` per server or peer process. The `Cache` extends `GemFireCache` and adds server/peer caching features like communication within the distributed system, region creation, transactions and querying, and cache server functionality and multi-site gateway senders and receivers for communication between distributed systems.
- **`com.gemstone.gemfire.cache.ClientCache`**. Use the `ClientCache` interface to manage the cache in your clients. You have one `ClientCache` per client process. The `ClientCache` extends

GemFireCache and adds client-specific caching features like client region creation, subscription keep-alive management for durable clients, querying on server and client tiers, and RegionService creation for secure access by multiple users within the client.

### The Cache XML

Your cache.xml must be formatted according to the dtd listed in the product dtd/cache7\_0.dtd.

You use one format for peer and server caches and another for client caches.

cache.xml for Peer/Server:

```
<?xml version="1.0"?>
<!DOCTYPE cache PUBLIC
  "-//GemStone Systems, Inc.//GemFire Declarative Caching 7.0//EN"
  "http://www.gemstone.com/dtd/cache7_0.dtd">
<cache>
  ...
</cache>
```

cache.xml for Client:

```
<?xml version="1.0"?>
<!DOCTYPE client-cache PUBLIC
  "-//GemStone Systems, Inc.//GemFire Declarative Caching 7.0//EN"
  "http://www.gemstone.com/dtd/cache7_0.dtd">
<client-cache>
  ...
</client-cache>
```

For more information on the cache.xml file, see [cache.xml](#) on page 861.

### Create and Close a Cache

Your system configuration and cache configuration are initialized when you start your member processes and create each member's GemFire cache.

The steps in this section use gemfire.properties and cache.xml file examples, except where API is required. You can configure your distributed system properties and cache through the API as well, and you can use a combination of file configuration and API configuration.

The XML examples listings may not include the full cache.xml file listing. All of your declarative cache configuration must conform to the cache DTD listing in the product installation dtd/cache7\_0.dtd.

For all of your GemFire applications:

1. Create your Cache, for peer/server applications, or ClientCache, for client applications. This connects to the GemFire system you have configured and initializes any configured data regions. Use your cache instance to access your regions and perform your application work.
2. Close your cache when you are done. This frees up resources and disconnects your application from the distributed system in an orderly manner.

Follow the instructions in the subtopics under [Cache Management](#) on page 137 to customize your cache creation and closure for your application needs. You may need to combine more than one of the sets of instructions. For example, to create a client cache in a system with security, you would follow the instructions for creating and closing a client cache and for creating and closing a cache in a secure system.

## Export and Import a Cache Snapshot

To aid in the administration of cache data and speed the setup of new environments, you can export a snapshot of the entire cache (all regions) and then import the snapshot into a new cache. For example, you could take a snapshot of the production environment cache in order to import the cache's data into a testing environment.

For more details on exporting and importing snapshots of a cache, see [Cache and Region Snapshots](#) on page 483.

## Managing a Peer or Server Cache

You start your peer or server cache using a combination of XML declarations and API calls. Close the cache when you are done.

GemFire peers are members of a GemFire distributed system that do not act as clients to another GemFire distributed system. GemFire servers are peers that also listen for and process client requests.

1. Create your cache:

- a. In your `cache.xml`, use the cache DOCTYPE and configure your cache inside a `<cache>` element. Example:

```
<!DOCTYPE cache PUBLIC
  "-//GemStone Systems, Inc.//GemFire Declarative Caching 7.0//EN"
  "http://www.gemstone.com/dtd/cache7_0.dtd">
<cache>
  // NOTE: Use this <cache-server> element only for server processes
  <cache-server port="40404"/>

  <region name="customerRegion" refid="REPLICATE" />
  <region name="ordersRegion" refid="PARTITION" />
</cache>
```

- b. Create the Cache instance:

- In your Java application, use the `CacheFactory.create` method:

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

- If you are running a server using the GemFire `cacheserver` process, it automatically creates the cache and connection at startup and closes both when it exits.

The system creates the distributed system connection and initializes the cache according to your `gemfire.properties` and `cache.xml` specifications.

2. Close your cache when you are done using the `close` method of your Cache instance:

```
cache.close();
```

## Managing a Client Cache

You have several options for client cache configuration. Start your client cache using a combination of XML declarations and API calls. Close the client cache when you are done.

GemFire clients are processes that send most or all of their data requests and updates to a GemFire server system. Clients run as standalone processes, without peers of their own.



**Note:** GemFire automatically configures the distributed system for your `ClientCache` as standalone, which means the client has no peers. Do not try to set the `gemfire.properties mcast-port` or `locators` for a client application or the system will throw an exception.

1. Create your client cache:

- a. In your `cache.xml`, use the `<client-cache>` DOCTYPE and configure your cache inside a `<client-cache>` element. Configure your server connection pool and your regions as needed. Example:

```
<!DOCTYPE client-cache PUBLIC "-//GemStone Systems, Inc./GemFire
Declarative Caching 7.0//EN"
"http://www.gemstone.com/dtd/cache7_0.dtd">
<client-cache>
  <pool name="serverPool">
    <locator host="host1" port="44444" />
  </pool>
  <region name="exampleRegion" refid="PROXY" />
</client-cache>
```



**Note:** Applications that use a `<client-cache>` may want to set `concurrency-checks-enabled` to false for a region in order to see all events for that region. GemFire server members can continue using concurrency checks, but they will pass all events to the client cache. This configuration ensures that the client sees all region events, but it does not prevent the client cache region from becoming out-of-sync with the server cache. See [Consistency for Region Updates](#) on page 239.

- b. If you use multiple server pools, configure the pool name explicitly for each client region. Example:

```
<pool name="svrPool1">
  <locator host="host1" port="40404" />
</pool>
<pool name="svrPool2">
  <locator host="host2" port="40404" />
</pool>
<region name="clientR1" refid="PROXY" pool-name="svrPool1"/>
<region name="clientR2" refid="PROXY" pool-name="svrPool2"/>
<region name="clientsPrivateR" refid="LOCAL"/>
```

- c. In your Java client application, create the cache using the `ClientCacheFactory.create` method. Example:

```
ClientCache clientCache = new ClientCacheFactory().create();
```

This creates the server connections and initializes the client's cache according to your `gemfire.properties` and `cache.xml` specifications.

2. Close your cache when you are done using the `close` method of your Cache instance:

```
cache.close();
```

If your client is durable and you want to maintain your durable queues while the client cache is closed, use:

```
clientCache.close(true);
```

## Managing a Cache in a Secure System

When you create your cache in a secure system, you provide credentials to the connection process for authentication by already-running, secure members. Clients connect to secure servers. Peers are authenticated by secure locators or peer members.

Follow these steps in addition to the steps for implementing security implementation for your peer, server, and client members. See [Managing a Peer or Server Cache](#) on page 139 and [Managing a Client Cache](#) on page 139.

1. To create your cache:

- a. Add any necessary security properties to the `gemfire.properties`, to configure vFabric GemFire for your particular security implementation. Examples:

```
security-client-auth-init=mySecurity.UserPasswordAuthInit.create
security-peer-auth-init=myAuthPkg.myAuthInitImpl.create
```

- b. When you create your cache, pass any properties required by your security implementation to the cache factory create call by using one of these methods:

- `ClientCacheFactory` or `CacheFactory` set methods. Example:

```
ClientCache clientCache = new ClientCacheFactory()
    .set("security-username", username)
    .set("security-password", password)
    .create();
```

- Properties object passed to the `ClientCacheFactory` or `CacheFactory` create method. These are usually properties of a sensitive nature that you do not want to put inside the `gemfire.properties` file. Example:

```
Properties properties = new Properties();
properties.setProperty("security-username", username);
properties.setProperty("security-password", password);
Cache cache = new CacheFactory(properties).create();
```



**Note:** Properties passed to a cache creation method override any settings in `gemfire.properties`.

2. Close your cache when you are done using the `close` method of your `Cache` or `ClientCache` instance. Example:

```
cache.close();
```

## Managing RegionServices for Multiple Secure Users

In a secure system, you can create clients with multiple, secure connections to the servers from each client. The most common use case is a GemFire client embedded in an application server that supports data requests from many users. Each user may be authorized to access a subset of data on the servers. For example, customer users may be allowed to see and update only their own orders and shipments.

In a single client, multiple authenticated users can all access the same `ClientCache` through instances of the `RegionService` interface. Because there are multiple users with varying authorization levels, access to cached data is done entirely through the servers, where each user's authorization can be managed.

Follow these steps in addition to the steps in [Managing a Cache in a Secure System](#) on page 140.

1. Create your cache and `RegionService` instances:

- a. Configure your client's server pool for multiple secure user authentication. Example:

```
<pool name="serverPool" multiuser-authentication="true">
  <locator host="host1" port="44444"/>
</pool>
```

This enables access through the pool for the `RegionService` instances and disables it for the `ClientCache` instance.

- b. After you create your `ClientCache`, from your `ClientCache` instance, for each user call the `createAuthenticatedView` method, providing the user's particular credentials. These are create method calls for two users:

```
Properties properties = new Properties();
properties.setProperty("security-username", cust1Name);
properties.setProperty("security-password", cust1Pwd);
RegionService regionService1 =
    clientCache.createAuthenticatedView(properties);

properties = new Properties();
properties.setProperty("security-username", cust2Name);
properties.setProperty("security-password", cust2Pwd);
RegionService regionService2 =
    clientCache.createAuthenticatedView(properties);
```

For each user, do all of your caching and region work through the assigned `RegionService` instance. Access to the server cache will be governed by the server's configured authorization rules for each individual user.

2. Close your cache by closing the `ClientCache` instance only. Do not close the `RegionService` instances first. This is especially important for durable clients.

### Requirements and Caveats for RegionService

Once each region is created, you can perform operations on it through the `ClientCache` instance or the `RegionService` instances, but not both.



**Note:** You can use the `ClientCache` to create a region that uses a pool configured for multi-user authentication, then access and do work on the region using your `RegionService` instances.

To use `RegionService`:

- Regions must be configured as `EMPTY`. Depending on your data access requirements, this configuration might affect performance, because the client goes to the server for every get.
- If you are running durable client queues (CQs) from the `RegionService` instances, stop and start the offline event storage for the client as a whole. The server manages one queue for the entire client process, so you need to request the stop and start of durable CQ event messaging for the cache as a whole, through the `ClientCache` instance. If you closed the `RegionService` instances, event processing would stop, but the server would continue to send events, and those events would be lost.

Stop with:

```
clientCache.close(true);
```

Start up again in this order:

1. Create `ClientCache` instance.
2. Create all `RegionService` instances. Initialize CQ listeners.
3. Call `ClientCache` instance `readyForEvents` method.

### Launching an Application after Initializing the Cache

You can specify a callback application that is launched after the cache initialization.

By specifying an `<initializer>` element in your `cache.xml` file, you can trigger a callback application, which is run after the cache has been initialized. Applications that use the `cacheserver` script to start up a server can also use this feature to hook into a callback application. To use this feature, you need to specify the callback class within the `<initializer>` element. This element should be added to the end of your `cache.xml` file.

You can specify the <initializer> element for either server caches or client caches.

The callback class must implement the Declarable interface. When the callback class is loaded, its `init` method is called, and any parameters defined in the <initializer> element are passed as properties.

The following is an example specification.

In cache.xml:

```
<initializer>
  <class-name>MyInitializer</class-name>
    <parameter name="members">
      <string>2</string>
    </parameter>
</initializer>
```

Here's the corresponding class definition:

```
import com.gemstone.gemfire.cache.Declarable;

public class MyInitializer implements Declarable {
  public void init(Properties properties) {
    System.out.println(properties.getProperty("members"));
  }
}
```

The following are some additional real-world usage scenarios:

1. Start a BridgeMembershipListener or a SystemMembershipListener

```
<initializer>
  <class-name>TestBridgeMembershipListener</class-name>
</initializer>
```

2. Write a custom tool that monitors cache resources

```
<initializer>
  <class-name>ResourceMonitorCacheXmlLoader</class-name>
</initializer>
```

Any singleton or timer task or thread can be instantiated and started using the initializer element.



# Chapter 21

---

## Data Regions

The region is the core building block of the vFabric GemFire distributed system. All cached data is organized into data regions and you do all of your data puts, gets, and querying activities against them.

### Data Region Management

GemFire provides different APIs and XML configuration models to support configuration and management of your data regions.

You store your data in region entry key/value pairs, with keys and values being any object types your application needs.

The `com.gemstone.gemfire.cache.Region` interface implements `java.util.Map`.

Each region's attributes define how the data in the region is stored, distributed, and managed. Data regions can be distributed, partitioned among system members, or local to the member.

You can create regions in the `cache.xml` file or by using the API. You can use *region shortcuts* to configure commonly-used types of regions. For more information about region shortcuts, see [Region Shortcuts Reference](#) on page 963.



**Note:** If you change attributes that define a region, you must restart the member before the changes take effect.

### The Region APIs

GemFire's regions APIs provide specialized behavior for different system member types.

- **Peer/Server Region APIs.** Use these methods, interfaces, and classes for peer/server region creation. These are in the `com.gemstone.gemfire.cache` package. They correspond to declarations in the `<cache>` element for creating and configuring regions.
  - **`com.gemstone.gemfire.cache.Cache.createRegionFactory`.** This method takes a `RegionShortcut` enum to initiate region configuration, and returns a `RegionFactory`.
  - **`com.gemstone.gemfire.cache.RegionFactory`.** Provides methods to set individual region attributes and to create the region. The `create` call returns `Region`.
  - **`com.gemstone.gemfire.cache.RegionShortcut`.** Common region configurations. Can be retrieved through `Cache.createRegionShortcut` and with the `region` attribute, `refid`.
- **Client Region APIs.** Use these methods, interfaces, and classes for client region creation. These are in the `com.gemstone.gemfire.cache.client` package. They correspond to declarations in the `<client-cache>` element for creating and configuring regions.

These are client versions of the Peer/Server Region APIs. These client APIs provide similar functionality, but are tailored to the needs and behaviors of client regions.

- **com.gemstone.gemfire.cache.clientCache.createRegionFactory**. This method takes a ClientRegionShortcut enum to initiate region configuration, and returns a ClientRegionFactory.
- **com.gemstone.gemfire.cache.client.ClientRegionFactory**. Provides methods to set individual region attributes and to create the region. The create call returns Region.
- **com.gemstone.gemfire.cache.client.ClientRegionShortcut**. Common region configurations. Can be retrieved through ClientCache createClientRegionFactory and with the region attribute, refid.
- **Region APIs Used For All Member Types**. These interfaces and classes are used universally for region management. These are in the com.gemstone.gemfire.cache package. They correspond to declarations under the <cache> and <client-cache> elements for creating and configuring regions.
  - **com.gemstone.gemfire.cache.Region**. Interface for managing your regions and their entries.
  - **com.gemstone.gemfire.cache.RegionAttributes**. Object holding region configuration settings.
  - **com.gemstone.gemfire.cache.AttributesFactory**. Can be used to create RegionAttributes to pass to RegionFactory and ClientRegionFactory.

## Create and Access Data Regions

Before you start, have your cache configuration defined, along with any cache-wide configuration your region requires, like disk store configuration and client server pool configuration.

1. Determine the region attributes requirements and identify the region shortcut setting that most closely matches your needs.
2. Define any region attributes that are not provided in the shortcut you chose.
3. Create a region using any of the following methods:

- Declaration in the cache.xml:

```
<?xml version="1.0"?>
<!DOCTYPE cache PUBLIC "-//GemStone Systems, Inc./GemFire Declarative
Caching 7.0//EN"
"http://www.gemstone.com/dtd/cache7_0.dtd">
<cache lock-lease="120" lock-timeout="60" search-timeout="300">
<!-- Create a region named Portfolios --&gt;
&lt;region name="Portfolios" id="REPLICATE"/&gt;
&lt;/cache&gt;</pre>
```

When the cache.xml is loaded at cache creation, the system automatically creates any declared regions.

- RegionFactory API calls:

```
RegionFactory rf = cache.createRegionFactory(REPLICATE);
Region pfloRegion = rf.create("Portfolios");
```

With RegionFactory, if the cache does not already exist, it is created, then the region is created. The RegionFactory constructor is overloaded to accept member connection properties and region attributes.

- Cache and Region API calls:

```
// Create a region
Cache cache = CacheFactory.create(system);
AttributesFactory factory = new AttributesFactory();
Region region = cache.createRegion("SampleRegion", factory.create());
```

Once you have created your regions, you can access them through the Cache and Region APIs as full region lists or individually.

## Update Data Regions

Update your region properties and contents through the API or from `cache.xml` file declarations.

- In the API, use Cache and Region methods to change configuration parameters and modify region structure and data.
- Load new XML declarations using the `Cache.loadCacheXml` method. Where possible, declarations in the new `cache.xml` file supersede existing definitions. For example, if a region declared in the `cache.xml` file already exists in the cache, its mutable attributes are modified according to the file declarations. Immutable attributes are not affected. If a region does not already exist, it is created. Entries and indexes are created or updated according to the state of the cache and the file declarations.

## Invalidate, Clear, and Destroy a Region

You can do these operations in two ways:

- As automated data expiration actions.
- As API calls, through the Region instance:

```
// Clear the local region
Region.localClear();

// Invalidate the entire distributed region
Region.InvalidateRegion();
```

You can remove all values or entries from the region or remove the region itself.

Operation type	Behavior
Region invalidate	Removes entry values from the region, leaving only data keys.
Region clear	Removes entry keys and values from the region, leaving the region empty of data.
Region destroy	Removes the region from the cache. For persistent regions, also removes data stored to disk.

Do this in the local cache only or as a distributed operation. The cache distributes the non-local operations according to the region's scope.



**Note:** Local invalidate and clear cannot be performed in replicated regions as doing so would invalidate the replication contract.

These operations cause event notification. The `CacheEvent` object includes the flag `getOperation.isExpiration` that is set to true for events resulting from expiration activities and to false for all others.

## Close a Region

Use this to stop local caching of persistent and partitioned regions without closing the entire cache:

```
region.close();
```

The `Region.close` operation works like the `Region.localDestroyRegion` operation with these significant differences:

- The `close` method is called for every callback installed on the region.
- No events are invoked. Of particular note, the entry events, `beforeDestroy` and `afterDestroy`, and the region events, `beforeRegionDestroy` and `afterRegionDestroy`, are not invoked. See [Events and Event Handling](#) on page 275.
- If persistent, the region is removed from memory but its disk files are retained.
- If partitioned, the region is removed from the local cache. If the partitioned region is redundant, local data caching fails over to another cache. Otherwise, local data is lost.

## Creating a Region with gfsh

A simple and fast way to create a data region in the vFabric GemFire cache is to use the `gfsh` command-line tool.

Before you start, configure your `Cache` or `ClientCache` and determine the region shortcut and other attributes settings your region needs.

Region creation is subject to attribute consistency checks, both internal to the cache and, if the region is not local, between all caches where the region is defined. The requirements for consistency between region attributes are detailed in the online Java API documentation.

1. Start a `gfsh` prompt.
2. Connect to a server that is currently acting as a JMX Manager node.
3. Enter the `create region` command providing any desired region attributes as arguments.

For example:

```
gfsh>create region --name=region1 --type=REPLICATE
```

See [create region](#) on page 679 for more details and examples.

4. Add data into the region by using the [import data](#) on page 696 command or [put](#) on page 704 `gfsh` command.
5. After you modify data in the region you can use the [export data](#) on page 691 command to generate a snapshot of the current region's data for later use.
6. Export the configuration files of your server so that you can save your region's configuration and recreate the region with the same attributes the next time you start up your cache server. See [export config](#) on page 690 for details.

## Creating a Region Through the cache.xml File

A common way to create a data region in the vFabric GemFire cache is through `cache.xml` declarations.

Before you start, configure your `<cache>` or `<client-cache>` in your `cache.xml` file and determine the region shortcut and other attributes settings your region needs. The `cache.xml` file must conform to the DTD in the product's `dtd/cache7_0.dtd`.

Region creation is subject to attribute consistency checks, both internal to the cache and, if the region is not local, between all caches where the region is defined. The requirements for consistency between region attributes are detailed in the online Java API documentation.

1. In your `cache.xml` file, create a `<region>` element for your new region as a subelement to the `<cache>` element or the `<client-cache>` element.
2. Define your region's name and a region attributes shortcut setting, if one applies. Find the shortcut setting that most closely fits your region configuration.
3. Add other attributes as needed to customize the region's behavior.

When you start your member with the `cache.xml` file, the region will be created.

## Examples

Partitioned Region Declaration

```
<region name="myRegion" refid="PARTITION" />
```

Partitioned Region Declaration with Backup to Disk

```
<region name="myRegion" refid="PARTITION_PERSISTENT" />
```

Partitioned Region Declaration with HA and Modified Storage Capacity in Host Member

```
<region name="myRegion" refid="PARTITION_REDUNDANT" >
  <region-attributes>
    <partition-attributes local-max-memory="512" />
  </region-attributes>
</region>
```

Replicated Region Declaration

```
<region name="myRegion" refid="REPLICATE" />
```

Replicated Region Declaration with Event Listener and Expiration

```
<region name="myRegion" refid="REPLICATE" >
  <region-attributes statistics-enabled="true" >
    <entry-time-to-live>
      <expiration-attributes timeout="60" action="destroy" />
    </entry-time-to-live>
    <cache-listener>
      <class-name>myPackage.MyCacheListener</class-name>
    </cache-listener>
  </region-attributes>
</region>
```

## Creating a Region Through the API

You can use the GemFire caching API to create regions in your cache after startup. For run-time region creation, you need to use the API.

Before you start, configure your Cache or ClientCache and determine the region shortcut and other attributes settings your region needs.

Region creation is subject to attribute consistency checks, both internal to the cache and, if the region is not local, between all caches where the region is defined. The requirements for consistency between region attributes are detailed in the online Java API documentation.

1. Use a region shortcut to create your region factory.
  - In peers and servers, use `com.gemstone.gemfire.cache.RegionFactory`.
  - In clients, use `com.gemstone.gemfire.cache.client.ClientRegionFactory`.
2. (Optional) Use the region factory to further configure your region.
3. Create your region from the configured region factory.

When you run your member with the region creation code, the region will be created in the cache.

### Examples

Creating a partitioned region using RegionFactory:

```
RegionFactory rf =
    cache.createRegionFactory(RegionShortcut.PARTITION);
rf.addCacheListener(new LoggingCacheListener());
custRegion = rf.create("customer");
```

Creating a modified partitioned region using RegionFactory:

```
PartitionAttributesFactory paf = new
PartitionAttributesFactory<CustomerId, String>();
paf.setPartitionResolver(new CustomerOrderResolver());

RegionFactory rf =
    cache.createRegionFactory(RegionShortcut.PARTITION);
rf.setPartitionAttributes(paf.create());
rf.addCacheListener(new LoggingCacheListener());
custRegion = rf.create("customer");
```

Creating a client region with a pool specification using ClientRegionFactory:

```
ClientRegionFactory<String, String> cRegionFactory =
    cache.createClientRegionFactory(PROXY);
Region<String, String> region =
    cRegionFactory.setPoolName("Pool3").create("DATA");
```

## Region Naming

To get the full range of vFabric GemFire capabilities for your cached data regions, follow GemFire's region naming guidelines.

The safest approach when naming your regions is to use only alphanumeric characters and the underscore character and to keep your region names short. This permits you the full range of GemFire region configurations and allow you to perform all available GemFire operations on the region.

Following these guidelines for naming regions:

- Do not use the separator slash character ‘ / ’.
- Do not begin region names with two underscore characters “\_\_” as this is reserved for internal GemFire use.
- To run queries against your data, restrict your region names to alphanumeric characters and the underscore character.

## Region Shortcuts and Custom Named Region Attributes

GemFire provides region shortcut settings, with preset region configurations for the most common region types. For the easiest configuration, start with a shortcut setting and customize as needed. You can also store your own custom configurations in the cache for use by multiple regions.

You configure automated management of data regions and their entries through region shortcuts and region attributes. These region configuration settings determine such things as where the data resides, how the region is managed in memory, membership roles and reliability behavior, and the automatic loading, distribution, and expiration of data entries.



**Note:** Whenever possible, use region shortcuts to configure your region, and further customize behavior using region attributes. The shortcut settings are preset with the most common region configurations.

GemFire provides a number of predefined, shortcut region attributes settings for your use. You can also define your own custom region attributes and store them with an identifier for later retrieval. Both types of stored

attributes are referred to as named region attributes. You can create and store your attribute settings in the `cache.xml` file and through the API.

Retrieve region shortcuts and custom named attributes by providing the ID to the region creation, in the `refid` attribute setting. This example uses the shortcut `REPLICATE` attributes to create a region:

```
<region name="testREP" refid="REPLICATE" />
```

You can create your own named attributes as needed, by providing an `id` in your region attributes declaration. The following region declaration:

1. Retrieves all of the attribute settings provided by the persistent partitioned region shortcut
2. Modifies the shortcut attribute settings by specifying a disk store name to use for persistence
3. Assigns the new attribute settings to the new region named `testPR`
4. Stores the attribute settings in a new custom attributes named `testPRPersist`:

```
<disk-store name="testDiskStore" >
  <disk-dirs>
    <disk-dir>PRPersist1</disk-dir>
    <disk-dir>PRPersist2</disk-dir>
  </disk-dirs>
</disk-store>
<region name="testPR" >
  <region-attributes id="testPRPersist"
    refid="PARTITION_PERSISTENT" disk-store-name="testDiskStore" />
</region>
```

## Shortcut Attribute Options

You can select the most common region attributes settings from GemFire's predefined named region attributes in these classes:

- `com.gemstone.gemfire.cache.RegionShortcut`. For peers and servers.
- `com.gemstone.gemfire.cache.client.ClientRegionShortcut`. For clients.

Shortcut attributes are a convenience only. They are just named attributes that GemFire has already stored for you. You can override their settings by storing new attributes with the same id as the predefined attributes.

The class Javadocs give complete listings of the options.

## RegionShortcuts for Peers and Servers

These are the primary options available in the region shortcut settings. The names listed appear in the shortcut identifier alone or in combination, like "PARTITION" in PARTITION, PARTITION\_PROXY, and PARTITION\_REDUNDANT.

### Cache Data Storage Model

- **PARTITION** . Creates a partitioned region. This is a data store for the region. You can also specify these options with PARTITION:
  - **PROXY** . Data is not stored in the local cache and the member is a data accessor to the region. This requires other members to create non-proxy copies of the region, so the data is stored somewhere.
  - **REDUNDANT** . The region stores a secondary copy of all data, for high availability.
- **REPLICATE** . Creates a replicated region. This is a data store for the region. You can also specify these options with REPLICATE:
  - **PROXY** . Data is not stored in the local cache and the member is a data accessor to the region. This requires other members to create non-proxy copies of the region, so the data is stored somewhere.

- **LOCAL** . Creates a region private to the defining member.

### Data Eviction

- **HEAP\_LRU** . Causes least recently used data to be evicted from memory when the GemFire resource manager determines that the cache has reached configured storage limits.

### Disk Storage

You can specify these alone or in combination:

- **PERSISTENT** . Backs up all data to disk, in addition to storing it in memory.
- **OVERFLOW** . Moves data out of memory and on to disk, when memory use becomes too high.

## ClientRegionShortcuts for Clients

These are the primary options available in the client region shortcut settings. The names listed appear in the shortcut identifier alone or in combination, like "PROXY" in PROXY and CACHING\_PROXY.

### Communication with Servers and Data Storage

- **PROXY** . Does not store data in the client cache, but connects the region to the servers for data requests and updates, interest registrations, and so on. The client is a data accessor to the region.
- **CACHING\_PROXY** . Stores data in the client cache and connects the region to the servers for data requests and updates, interest registrations, and so on.
- **LOCAL** . Stores data in the client cache and does not connect the region to the servers. This is a client-side-only region. Note that this is not the same as setting the region's scope attribute to LOCAL.

### Data Eviction

- **HEAP\_LRU** . Causes least recently used data to be evicted from memory when the GemFire resource manager determines that the cache has reached configured storage limits.

### Disk Storage

With the LOCAL and CACHING data storage shortcut options, you can also specify these disk storage options, alone or in combination:

- **PERSISTENT** . Backs up all data to disk, in addition to storing it in memory.
- **OVERFLOW** . Moves data out of memory and on to disk, when memory use becomes too high.

## Storing and Retrieving Region Shortcuts and Custom Named Region Attributes

Use these examples to get started with GemFire region shortcuts.

GemFire region shortcuts, in com.gemstone.gemfire.cache.RegionShortcut for peers and servers and com.gemstone.gemfire.cache.client.ClientRegionShortcut for clients, are available wherever you create a region in the cache.xml or through the API. Custom named attributes, stored by you, are available from the moment you store them on.

The region shortcuts are special GemFire named region attributes, with identifying names. Create custom named region attributes by setting the attributes and storing them with a unique identifier in the region attribute id. Retrieve named attributes by providing the shortcut enum value or the name you assigned in the id to the region creation:

- In the API, use the identifier in the region factory creation
- In the cache.xml, use the identifier in the <region> or <region-attribute> refid setting. The refid is available in both elements for convenience

## Examples

### Example #1

This example shows partitioned region creation in the `cache.xml`:

- The first `region-attributes` declaration starts with the predefined `PARTITION_REDUNDANT` attributes, modifies the `local-max-memory` setting, and stores the resulting attributes in the custom-named `myPartition` attributes.
- The region declarations use the new stored attributes, but each has its own interest policy, which is specified in the individual region creation.

```
<!-- Retrieving and storing attributes -->
<region-attributes id="myPartition" refid="PARTITION_REDUNDANT">
  <partition-attributes local-max-memory="512"/>
</region-attributes>

<!-- Two partitioned regions, one colocated with the other -->

<!-- Attributes are retrieved and applied in the first region -->
<region name="PartitionedRegion1" refid="myPartition"/>

<!-- Same stored attributes, modification for this region-->
<region name="PartitionedRegion2" refid="myPartition">
  <region-attributes>
    <partition-attributes colocated-with="PartitionedRegion1" />
  </region-attributes>
</region>
```

### Example #2

This example uses the `RegionFactory` API to create a region based on the predefined `PARTITION` region shortcut:

```
final Region diskPortfolios =
  new RegionFactory("PARTITION").create("Portfolios");
```

This example retrieves an attributes template and passes it to the region creation with a modified pool specification:

```
ClientRegionFactory<String, String> regionFactory =
  cache.createClientRegionFactory(PROXY);
Region<String, String> region = regionFactory
  .setPoolName("publisher")
  .create("DATA");
```

## Managing Region Attributes

Use region attributes to fine-tune the region configuration provided by the region shortcut settings.

All region attributes have default settings, so you only need to use region attributes to set the ones you want to override. See [<region-attributes>](#) on page 882.

### Define Region Attributes

Create region attributes using any of these methods:

- Declarations inside the `cache.xml` `<region>` element:

```
<cache>
  <region name="exampleRegion" refid="REPLICATE">
    <region-attributes statistics-enabled="true">
      <entry-idle-time>
        <expiration-attributes timeout="10" action="destroy"/>
      </entry-idle-time>
      <cache-listener>
        <class-name>quickstart.SimpleCacheListener</class-name>
      </cache-listener>
    </region-attributes>
  </region>
</cache>
```

```

        </cache-listener>
    </region-attributes>
</region>
</cache>
```

When the `cache.xml` is loaded at startup, declared region attributes are automatically created and applied to the region.

- RegionFactory API set\* method calls:

```

// Creating a partitioned region using the RegionFactory
RegionFactory rf = cache.createRegionFactory(RegionShortcut.PARTITION);
rf.addCacheListener(new LoggingCacheListener());
custRegion = rf.create("customer");

// Creating a partitioned region using the RegionFactory, with attribute
modifications
PartitionAttributesFactory paf =
    new PartitionAttributesFactory<CustomerId, String>();
paf.setPartitionResolver(new CustomerOrderResolver());

RegionFactory rf =
    cache.createRegionFactory(RegionShortcut.PARTITION);
rf.setPartitionAttributes(paf.create());
rf.addCacheListener(new LoggingCacheListener());
custRegion = rf.create("customer");

// Creating a client with a Pool Specification Using ClientRegionFactory
ClientRegionFactory<String, String> cRegionFactory =
    cache.createClientRegionFactory(PROXY);
Region<String, String> region =
    cRegionFactory.setPoolName("Pool3").create("DATA");
```

- AttributesFactory API:

1. Create an AttributesFactory instance
2. Use the instance to set non-default attributes
3. Run the factory create method to create a RegionAttributes instance
4. Pass the RegionAttributes instance to the region create method:

```

Cache cache = CacheFactory.create(system);
AttributesFactory factory = new AttributesFactory();
factory.setScope(Scope.DISTRIBUTED_ACK);
factory.setDataPolicy(DataPolicy.REPLICATE);
Region region = cache.createRegion("SampleRegion", factory.create());

// Create a new region using another region's attributes
AttributesFactory fac =
    new AttributesFactory(this.currRegion.getAttributes());
Region nr = cache.createRegion(name, fac.create());
```

## Modify Region Attributes

You can modify a region's event handlers and expiration and eviction attributes after the region is created.



**Note:** Do not modify attributes for existing regions unless you have to. Creating the attributes you need at region creation is more efficient.

Modify attributes in one of these ways:

- By loading a `cache.xml` with modified region attribute specifications:

```
<!-- Change the listener for exampleRegion
...
<region name="exampleRegion">
    <region-attributes statistics-enabled="true">
        <cache-listener>
            <class-name>quickstart.ComplexCacheListener</class-name>
        </cache-listener>
    </region-attributes>
</region>
...
```

- Using the `AttributesMutator` API:

1. Retrieve the `AttributesMutator` from the region
2. Call the mutator set methods to modify attributes:

```
currRegion = cache.getRegion("root");
AttributesMutator mutator = this.currRegion.getAttributesMutator();
mutator.addCacheListener(new LoggingCacheListener());
```

## Creating Custom Attributes for Regions and Entries

Use custom attributes to store information related to your region or its entries in your cache. These attributes are only visible to the local application and are not distributed.

You can define custom user attributes so you can associate data with the region or entry and retrieve it later. Unlike the other configuration settings, these attributes are used only by your application.



**Note:** User attributes are not distributed.

1. Create a Java Object with your attribute definitions.
2. Attach the object to the region or to an entry:

- `Region.setUserAttribute(userAttributeObject)`
- `Region.getEntry(key).setUserAttribute(userAttributeObject)`

3. Get the attribute value:

- `Region.getUserAttribute()`
- `Region.getEntry(key).getUserAttribute()`

This example stores attributes for later retrieval by a cache writer.

```
// Attach a user attribute to a Region with database info for table portfolio
Object myAttribute = "portfolio";
final Region portfolios =
    new RegionFactory().setCacheWriter(new
PortfolioDBWriter()).create("Portfolios");
Portfolios.setUserAttribute(myAttribute);

// Implement a cache writer that reads the user attribute setting
public class PortfolioDBWriter extends CacheWriterAdapter {
    public void beforeCreate(RegionEvent event) {
        table = (String)event.getRegion().getUserAttribute();
        // update database table using name from attribute
        . . .
    }
}
```

### **Limitations and Alternatives**

User attributes are not distributed to other processes, so if you need to define each attribute in every process that uses the region or entry. You need to update every instance of the region separately. User attributes are not stored to disk for region persistence or overflow, so they cannot be recovered to reinitialize the region.

If your application requires features not supported by user attributes, an alternative is to create a separate region to hold this data instead. For instance, a region, AttributesRegion, defined by you, could use region names as keys and the user attributes as values. Changes to AttributesRegion would be distributed to other processes, and you could configure the region for persistence or overflow if needed.

# Chapter 22

---

## Data Entries

The data entry is the key/value pair where you store your data. You can manage your entries individually and in batches. To use domain objects for your entry values and keys, you need to follow vFabric GemFire requirements for data storage and distribution.

### Managing Data Entries

Program your applications to create, modify, and manage your cached data entries.



**Note:** If you do not have the `cache copy-on-read` attribute set to true, do not change the objects returned from the Java entry access methods. Instead, create a copy of the object, then modify the copy and pass it to the Java `put` method. Modifying a value in place bypasses the entire distribution framework provided by GemFire, including cache listeners and expiration activities, and can produce undesired results.

#### Basic Create and Update

To create or update an entry in the cache, use `Region.put`. For example:

```
String name = ...  
String value = ...  
this.currRegion.put(name,value);
```



**Note:** You can also use the `gfsh put` command to add an entry to a region. See [put](#) on page 704 for more information.

If you want only to create the entry (with a null value and with method failure if the entry already exists), use `Region.create` instead.

#### Batch Entry Gets and Puts

To batch updates to multiple entries, use `Region.putAll` and `Region.getAll`. The method takes a Map of key-value pairs and puts them into the cache and distributes them in a single operation.

```
void putAll(String command) throws CacheException  
{  
    // Get Entry keys and values into Strings key1, ... keyN and value1, ...  
    valueN  
    Map map = new LinkedHashMap();  
    map.put(key1, value1));  
    ...  
    map.put(keyN, valueN));
```

```
    this.currRegion.putAll(map);  
}
```

The updates to the cache are done individually in the order in which they were placed in the Map. For partitioned regions, multiple events are sent as a single message to the primary buckets and then distributed to the secondary buckets.



**Note:** The processing of maps with very many entries and/or very large data may affect system performance and cause cache update timeouts, especially if the region uses overflow or persistence to disk.

### Safe Entry Modification

When you get an entry value from the cache, by default, the retrieval methods return a direct reference to the cached object. This provides the value as quickly as possible, but also opens the cache to direct, in-place changes.



**Note:** Do not directly modify cached values. Modifying a value in place bypasses the GemFire distribution framework, including cache writers and listeners, expiration activities, and transaction management, and can produce undesired results.

Always change your entries using copies of the retrieved objects—never directly modify the returned objects. You can do this in one of two ways:

1. Change the entry retrieval behavior for your cache by setting the cache attribute, `copy-on-read`, to true (the default is false).

```
<cache copy-on-read="true">  
...  
</cache>
```

When `copy-on-read` is true, the entry access methods return copies of the entries. This protects you from inadvertently modifying in-place, but negatively impacts performance and memory consumption when copying is not needed.

These entry access methods return an entry reference if `copy-on-read` is false and a copy of the entry if `copy-on-read` is true:

<code>Region.get</code>	<code>result of Region.put</code>	<code>EntryEvent.getNewValue</code>
<code>Region.values</code>	<code>Region.Entry.getValue</code>	<code>EntryEvent.getOldValue</code>
<code>Query.select</code>		

2. Create a copy of the returned object and work with that. For objects that are cloneable or serializable, you can copy the entry value to a new object using `com.gemstone.gemfire.CopyHelper.copy`. Example:

```
Object o = (StringBuffer)region.get("stringBuf");  
StringBuffer s = (StringBuffer) CopyHelper.copy(o);  
s.append("Changes to value, added using put.");  
region.put("stringBuf", s);
```

### Retrieving Region Entries from Proxy Members

The `Region.values` method call applies to the local region instance only. If you call the `values` method from a client region using the PROXY shortcut, the method call will not be redirected to the server region. To obtain a collection of all values in the Region from a client, you should use interest registration on `ALL_KEYS`, or use a query.

If you use the `Region.get` method from a proxy member, the method call will redirect to the region on the server if it cannot find the key locally.

## Requirements for Using Custom Classes in Data Caching

Follow these guidelines to use custom domain classes for your cached entry keys and values.

### CLASSPATH

Each member's CLASSPATH must include classes for all objects the member accesses.

- For Java applications, use the standard Java CLASSPATH.
- For the cacheserver process, use the CLASSPATH environment variable or the cacheserver's `-classpath` parameter. See [vFabric GemFire Servers](#) on page 741.

Data is sent between clients and servers in serialized form and the server stores client data in serialized form. The server does not need to deserialize data to send it to another client or to access it through a `PDXInstance`, but it does need to deserialize it to access it in other ways. The server CLASSPATH must include the classes for:

- All entry keys
- Entry values in regions that the server persists to disk
- Entry values the server accesses for any reason other than access using a `PdxInstance` or transfer of the full entry value to a client

For information on `PdxInstances`, see [Data Serialization](#) on page 257.

### Data Serialization

GemFire serializes data entry keys and values for distribution, so all data that GemFire moves out of the local cache for any reason must be serializable. Additionally, partitioned regions store data in serialized form. Almost every configuration requires serialization.

For information on the requirements and options for data serialization, see [Data Serialization](#) on page 257.

### Classes Used as Keys

The region uses hashing on keys. If you define a custom class to use as a key, for the class, override:

- `equals`
- `hashCode`. The default `hashCode` inherited from `Object` uses identity, which is different in every system member. In partitioned regions, hashing based on identity puts data in the wrong place. For details, see the Java API documentation for `java.lang.Object`.



# Part 5

---

## Topologies and Communication

*Topologies and Communication* explains how to plan and configure vFabric GemFire member discovery, peer-to-peer and client/server communication, and multi-site topologies.

### Topics:

- [Topology and Communication General Concepts](#)
- [Peer-to-Peer Configuration](#)
- [Client/Server Configuration](#)
- [Multi-site \(WAN\) Configuration](#)



# Chapter 23

---

## Topology and Communication General Concepts

Before you configure your vFabric GemFire members, make sure you understand the options for topology and communication.

### Topology Types

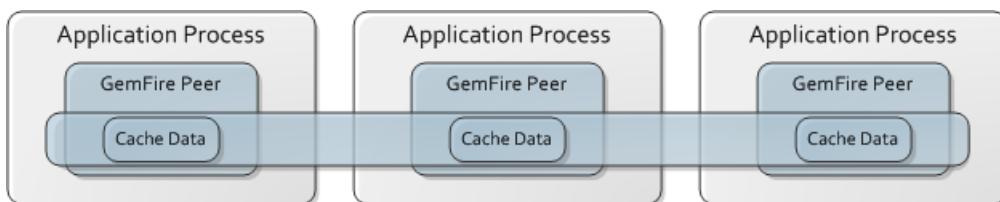
The vFabric GemFire topology options allow you to scale horizontally and vertically.

vFabric GemFire provides a variety of cache topologies:

- At the core of all systems is the single, peer-to-peer distributed system.
- For horizontal and vertical scaling, you can combine individual systems into client/server and multi-site (WAN) topologies:
  - In client/server systems, a small number of server processes manage data and event processing for a much larger client group.
  - In multi-site systems, several geographically disparate systems are loosely coupled into a single, cohesive processing unit.

### Peer-to-Peer Configuration

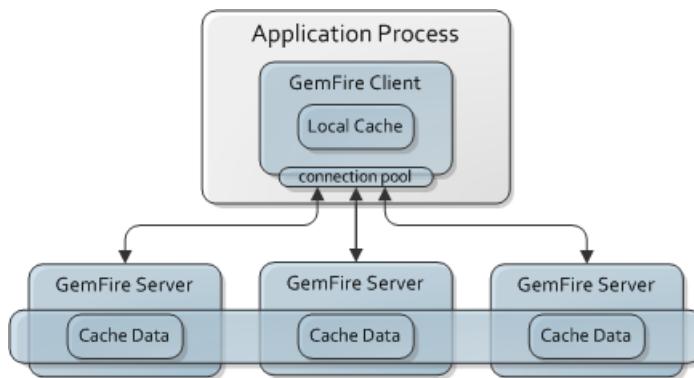
The peer-to-peer distributed system is the building block for all GemFire installations. Peer-to-peer alone is the simplest topology. Each cache instance, or member, directly communicates with each every other member in the distributed system. This cache configuration is primarily designed for applications that need to embed a cache within the application process space and participate in a cluster. A typical example is an application server cluster in which the application and the cache are co-located and share the same heap.



### Client/Server Configuration

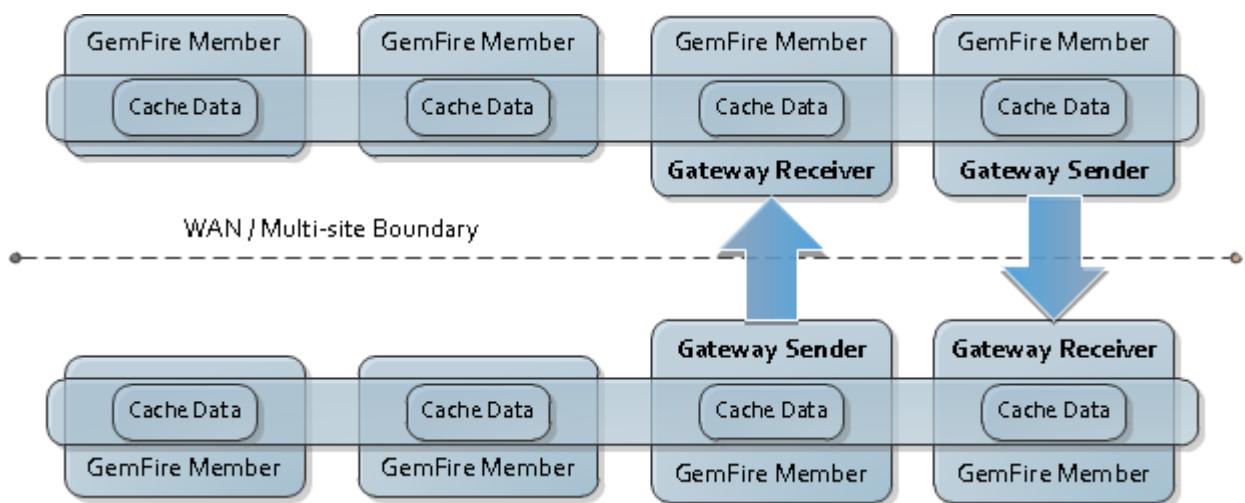
The client/server topology is the model for vertical scaling, where clients typically host a small subset of the data in the application process space and delegate to the server system for the rest. Compared to peer-to-peer by

itself, the client/server architecture provides better data isolation, high fetch performance, and more scalability. If data distribution will put a very heavy load on the network, a client/server architecture usually gives better performance. In any client/server installation, the server system is itself a peer-to-peer system, with data distributed between servers. A client system has a connection pool, which it uses to communicate with servers and other GemFire members. A client may also contain a local cache.



### Multi-site Configuration

For horizontal scaling, you can use a loosely coupled multi-site topology. With multi-site, multiple GemFire systems are loosely coupled, generally across geographical distances with slower connections, such as with a WAN. This topology provides better performance than the tight coupling of a single system, and greater independence between locations, so that each site can function on its own if the connection or remote site become unavailable. In a multi-site installation, each individual site is a peer-to-peer or Client/Server system.



### Planning Topology and Communication

Create a topology plan and a detailed list of machines and communication ports that your members will use. Configure your vFabric GemFire systems and configure the communication between the systems.

Your configuration governs how your applications find each other and distribute events and data among themselves.

1. Work with your system administrator to determine the protocols and addresses you will use for membership and communication. The communication details you determine in this step will be used in the system configuration steps that follow.
  - a. For each host machine with more than one network adapter card, decide whether to use the default address or one or more non-default bind addresses. You can use different cards for peer, server, and WAN connections.
  - b. Identify any members you want to run as standalone, isolated members, with no member discovery. This is sometimes a good option for clients. It has faster startup, but no peer-to-peer distribution of any kind.
  - c. For all non-standalone members:
    - Decide whether you will use GemFire locators for member discovery. They are recommended for production systems and required for implementing security, network partitioning management, and client/server installations. Decide how many locators you will use and where they will run. To ensure the most stable startup and availability, use multiple locators run on multiple machines. Create a list of your locators' address and port pairs. You will use the list to configure your system members, any clients, and the locators themselves.
    - If you will use multicasting for communication or as an alternate to locators for peer member discovery, note the addresses and ports. Select both unique multicast ports and unique addresses for your distributed systems.



**Note:** Use different port numbers for different systems, even if you use different multicast addresses. Some operating systems do not keep communication separate between systems that have unique addresses but the same port number.

2. Set up membership in your systems. See [Configuring Peer-to-Peer Discovery](#) on page 173.
3. Set up communication between system members. See [Configuring Peer Communication](#) on page 174.
4. As needed, set up communication between your systems. See [Configuring a Client/Server System](#) on page 183 and [Configuring a Multi-site \(WAN\) System](#) on page 195.

When you run your systems, your members will find each other and communicate using the configured methods and addresses.

## How Member Discovery Works

vFabric GemFire provides various options for member discovery within a distributed system and between clients and servers.

- [Peer Member Discovery](#) on page 165
- [Standalone Member](#) on page 166
- [Client Discovery of Servers](#) on page 166
- [Multi-site Discovery](#) on page 167

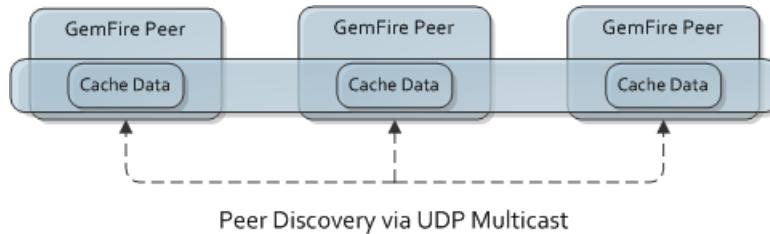
### Peer Member Discovery

Peer 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. Once they have found each other, members communicate directly, independent of the discovery mechanism. In peer discovery, GemFire uses a membership coordinator to manage member joins and departures.

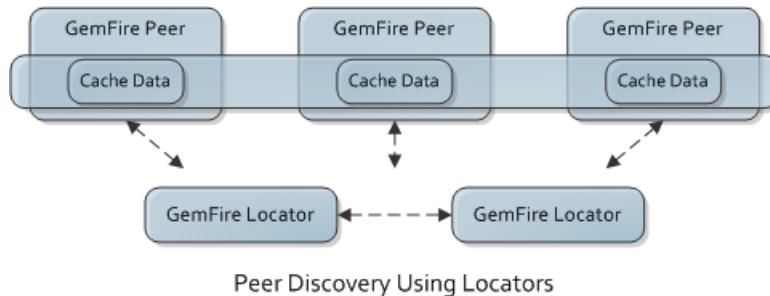
There are two discovery options: using multicast or using locators. By default, GemFire peers discover each other using multicast communication on a known port. Alternatively, you can configure member discovery using one or more dedicated locators. A locator provides both discovery and load balancing services.

- **UDP/IP Multicast.** New members broadcast their connection information over the multicast address and port to all running members. Existing members respond to establish communication with the new member. By default, peers discover each other using multicast communication.

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



- **GemFire Locators Using TCP/IP.** Peer locators manage a dynamic list of distributed system members. New members connect to one of the locators to retrieve the member list, which it uses to join the system.



 **Note:** Locators are recommended for discovery in all production systems. Multiple locators ensure the most stable startup and availability for your distributed system. You must use peer locators if you implement security and authorization or network partitioning management. You must use locators if you use a multi-site (WAN) configuration.

Locators are given preference over multicasting for member discovery. If you have both peer locators and multicast configured, the locators are used.

### Standalone Member

The standalone member has no peers, does no peer discovery, and so does not use locators or multicasting. It creates a distributed system connection only to access the GemFire caching features. 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 members can be accessed and monitored if you enable the member to become a JMX Manager.

### Client Discovery of Servers

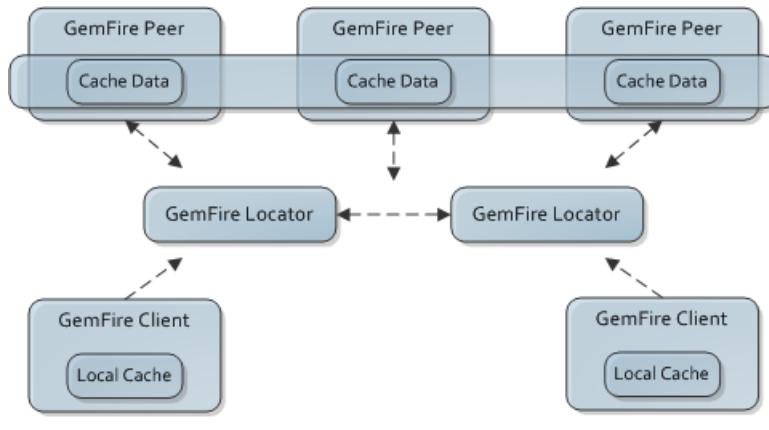
Locators provide clients with dynamic server discovery and server load balancing. Clients are configured with locator information for the server system, and turn to the locators for directions to the servers to use. The servers can come and go and their capacity to service new client connections can 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.



**Note:** For performance and cache coherency, clients must run as standalone members or in different distributed systems than their servers.

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 is the standard configuration.



Client/Server Discovery Using Locators

## Multi-site Discovery

In a multi-site (WAN) configuration, a GemFire cluster uses locators to discover remote GemFire clusters as well as to discover local GemFire members. Each locator in a WAN configuration uniquely identifies the local cluster to which it belongs, and it can also identify locators in remote GemFire clusters to which it will connect for WAN distribution.

When a locator starts up, it contacts each remote locator to exchange information about the available locators and gateway receiver configurations in the remote cluster. In addition to sharing information about its own cluster, a locator shares information that it has obtained from all other connected clusters. Each time a new locator starts up or an existing locator shuts down, the changed information is broadcast to other connected GemFire clusters across the WAN.

See [Discovery for Multi-Site Systems](#) on page 190 for more information.



**Note:** When you configure a multi-site system, you *must* use locators for discovery rather than multicast.

## How Communication Works

GemFire uses a combination of TCP unicast and UDP unicast and multicast for communication between members. You can change the default behavior to optimize communication for your system.

Client/server communication and gateway sender to gateway receiver communication uses TCP/IP sockets. The server listens for client communication at a published address and the client establishes the connection, sending its location. Similarly, the gateway receiver listens for gateway sender communication and the connection is established between sites.

In peer systems, for general messaging and region operations distribution, GemFire uses either TCP or UDP unicast. The default is TCP. You can use TCP or UDP unicast for all communications or you can use it as the default but then can target specific regions to use UDP multicast for operations distribution. The best combination for your installation depends in large part on your data use and event messaging.

### TCP Unicast

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 and its 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.



**Note:** Even when GemFire is configured to use UDP for messaging, GemFire uses a TCP connection when attempting to detect failed members. See [Failure Detection and Membership Views](#) on page 490 for more details.

### UDP Unicast and Multicast

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. 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.

For each member, GemFire selects a unique port for UDP unicast communication. You can restrict the range used for the selection by setting `membership-port-range` in the `gemfire.properties` file. Example:  
`membership-port-range=1024-60000`



**Note:** In addition to UDP port configuration, the `membership-port-range` property defines the TCP port used for failure detection. See [gemfire.properties and gfsecurity.properties \(GemFire Properties\)](#) on page 851 for a description of the GemFire property.

#### 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.

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.

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.

Even if you use multicast for a region, GemFire will send 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.

## Using Bind Addresses

You use a bind address configuration to send network traffic through non-default network cards and to distribute the load of network traffic for GemFire across multiple cards. If no bind address setting is found, GemFire uses the host machine's default address.

Host machines transmit data to the network and receive data from the network through one or more network cards, also referred to as network interface cards (NIC) or LAN cards. A host with more than one card is referred to as a multi-homed host. On multi-homed hosts, one network card is used by default. You can use bind addresses to configure your GemFire members to use non-default network cards on a multi-homed host.



**Note:** When you specify a non-default card address for a process, all processes that connect to it need to use the same address in their connection settings. For example, if you use bind addresses for your server locators, you must use the same addresses to configure the server pools in your clients.

Use IPv4 or IPv6 numeric address specifications for your bind address settings. For information on these specifications, see [Choosing Between IPv4 and IPv6](#) on page 171. Do not use host names for your address specifications. Host names resolve to default machine addresses.

## Peer, Server, and Gateway Receiver Communication

You can configure peer, server, and gateway receiver communication so two or more types use the same address or so each communication type uses its own address. If no setting is found for a specific communication type, GemFire uses the host machine's default address.



**Note:** Bind addresses set through the APIs, like CacheServer and DistributedSystem, take precedence over the settings discussed here. If your settings are not working, check to make sure there are no bind address settings being done through API calls.

This table lists the settings used for peer, server, and gateway receiver communication, ordered by precedence. For example, for gateway receiver communication, GemFire searches first for a `cache.xml <gateway-receiver> bind-address` setting. If that is not set, GemFire searches for the `gfsh start server server-bind-address` setting, and so on until a setting is found or all possibilities are exhausted.

Property Setting Ordered by Precedence	Peer	Server	Gateway Receiver	Syntax
<code>cache.xml &lt;gateway-receiver&gt; bind-address</code>			X	<code>&lt;gateway-receiver&gt;bind-address=address</code>
<code>cache.xml &lt;cache-server&gt; bind-address</code>		X		<code>&lt;cache-server&gt;bind-address=address</code>
<code>gfsh start server command-line --server-bind-address</code>		X	X	<code>gfsh start server --server-bind-address=address</code>

Property Setting Ordered by Precedence	Peer	Server	Gateway Receiver	Syntax
gemfire.properties server-bind-address		X	X	server-bind-address= <i>address</i>
gemfire.properties bind-address	X	X	X	bind-address= <i>address</i>

For example, a member started with these configurations in its `gemfire.properties` and `cache.xml` files will use three separate addresses for peer, server, and gateway receiver communication:

```
// gemfire.properties setting for peer communication
bind-address=194.124.0.104

//cache.xml settings
<cache>
  // Gateway receiver configuration
    <gateway-receiver start-port="1530" end-port="1551"
    bind-address="194.124.0.105"/>

  // Server communication
    <cache-server bind-address="194.124.0.106" ...>

  <region ...>
```

#### Related Topics

[cache.xml](#) on page 861

[vFabric GemFire Servers](#) on page 741

[gemfire.properties and gfsecurity.properties \(GemFire Properties\)](#) on page 851

## Locator Communication

Set the locator bind address using one of these methods:

- On the gfsh command line, specify the bind address when you start the locator, the same as you specify any other address:

```
gfsh>start locator --name=my_locator --bind-address=ip-address-to-bind
--port=portNumber
```

- Inside a GemFire application, take one of the following actions:
  - Automatically start a co-located locator using the `gemfire` property `start-locator`, and specifying the bind address for it in that property setting.
  - Use `com.gemstone.gemfire.distributed.LocatorLauncher` API to start the locator inside your code. Use the `LocatorLauncher.Builder` class to construct an instance of the `LocatorLauncher`, use the `setBindAddress` method to specify the IP address to use and then use the `start()` method to start a Locator service embedded in your Java application process.

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 locator's bind address and the locator's port in your `gemfire.properties` `locators` list. For server discovery in a client/server installation, use the locator's bind address and the locator's port in the locator list you provide to in the client's server pool configuration.

#### Related Topics

[vFabric GemFire Locator Process](#) on page 737

## Choosing Between IPv4 and IPv6

By default, vFabric GemFire uses Internet Protocol version 4 for GemFire address specifications.

You can switch to Internet Protocol version 6 if all your machines support it. You may lose performance, so you need to understand the costs of making the switch.

- IPv4 uses a 32-bit address. 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 uses a 128-bit address. IPv6 succeeds IPv4, and will provide a much greater number of addresses.

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.



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

IPv4 is the default version.

To use IPv6, set the Java property, `java.net.preferIPv6Addresses`, to `true`.

These examples show the formats to use to specify addresses in GemFire.

- IPv4:

```
239.192.81.2
```

- IPv6:

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



# Chapter 24

---

## Peer-to-Peer Configuration

Use peer-to-peer configuration to set member discovery and communication within a single distributed system.

### Configuring Peer-to-Peer Discovery

By default, vFabric GemFire uses multicast for peer discovery. You can change the configuration to separate distributed systems and provide greater flexibility in member discovery.

Before you begin, you should have a basic understanding of member discovery. You should also have already determined the address and port settings for member discovery. See [Topology and Communication General Concepts](#) on page 163.

These options are set through GemFire properties. See [gemfire.properties and gfsecurity.properties \(GemFire Properties\)](#) on page 851.

- To use **multicast** for membership, in the `gemfire.properties` for all system peers, provide the multicast address and port and disable locators for peer discovery:

```
locators=
mcast-address=<mcast address>
mcast-port=<mcast port>
```

- To use **locators** for membership, in the `gemfire.properties` of all system peers, including the locators, provide the locator list:

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

By default, locators provide peer discovery in the system in which they run, so no special settings are required to enable it when you start them. See [vFabric GemFire Locator Process](#) on page 737.

- To use locators in a WAN configuration, specify a unique distributed system ID for the local cluster, as well as the locator(s) available in remote clusters to which you will connect. For example:

```
mcast-port=0
locators=<locator1-address>[<port1>],<locator2-address>[<port2>]
distributed-system-id=1
remote-locators=<remote-locator-addr1>[<port1>],<remote-locator-addr2>[<port2>]
```

- To run a standalone member, in the `gemfire.properties` disable multicasting and locators:

```
locators=
mcast-address=
mcast-port=0
```



**Note:** Multicast and locator settings must be consistent throughout the distributed system.

## Configuring Peer Communication

By default vFabric GemFire uses TCP for communication between members of a single distributed system. You can modify this at the member and region levels.

Before you begin, you should have a basic understanding of member discovery. You should also have already determined the address and port settings for multicasting, including any bind addresses. See [Topology and Communication General Concepts](#) on page 163.

See [gemfire.properties](#) and [gfsecurity.properties](#) (GemFire Properties) on page 851 and [cache.xml](#) on page 861.

- Configure general messaging to use TCP or UDP unicast.**

TCP is the default protocol for communication. To use it, just make sure you do not have it disabled in `gemfire.properties`. Either have no entry for `disable-tcp`, or have this entry:

```
disable-tcp=false
```

To use UDP unicast for general messaging, add this entry to `gemfire.properties`:

```
disable-tcp=true
```

The `disable-tcp` setting has no effect on the use of TCP locators.

- Configure any regions you want to distribute using UDP multicast.**

- Configure UDP multicast for region messaging, set non-default multicast address and port selections in `gemfire.properties`:

```
mcast-address=<address>
mcast-port=<port>
```

- In `cache.xml`, enable multicast for each region that needs multicast messaging:

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



**Note:** 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. In addition, you may need to address interrelated setup and tuning issues at the GemFire, operating system, and network level.

Once your members establish their connections to each other, they will send distributed data and messages according to your configuration.

## Organizing Peers into Logical Member Groups

In a peer-to-peer configuration, you can organize members into logical member groups and use those groups to associate specific data or assign tasks to a pre-defined set of members.

You can use logical member groups to deploy JAR applications across multiple members or to execute functions across a member group.

To add a peer to a member group, you can configure the following:

- Add the member group names to the `gemfire.properties` file for the member. For example:

```
#gemfire.properties
groups=Portfolios,ManagementGroup1
```

A member can belong to more than one member group. If specifying multiple member groups for a member, use a comma-separated list. Alternatively, if you are using the `gfsh` command interface to start up the member, provide the group name or group names as a parameter.

For example, to start up a server and associate it with member groups, you could type:

```
gfsh>start server --name=server1 \
--group=Portfolios,MgmtGroup1,GatewaySenders
```

For example, to start up a locator and associate it with member groups, you could type:

```
gfsh>start locator --name=locator1 \
--group=ManagementGroup1
```

2. Then you can use the member group names to perform tasks such as starting gateway senders or gateway receivers, deploy applications or execute functions.

For example, to start a gateway sender on all members in a member group, you could type the following in gfsh:

```
gfsh>start gateway-sender --id=sender1 \
--group=GatewaySenders
```

For example, to deploy an application across a member group, you could type the following in gfsh:

```
gfsh>deploy --jar=group1_functions.jar --group=MgmtGroup1
```



# Chapter 25

---

## Client/Server Configuration

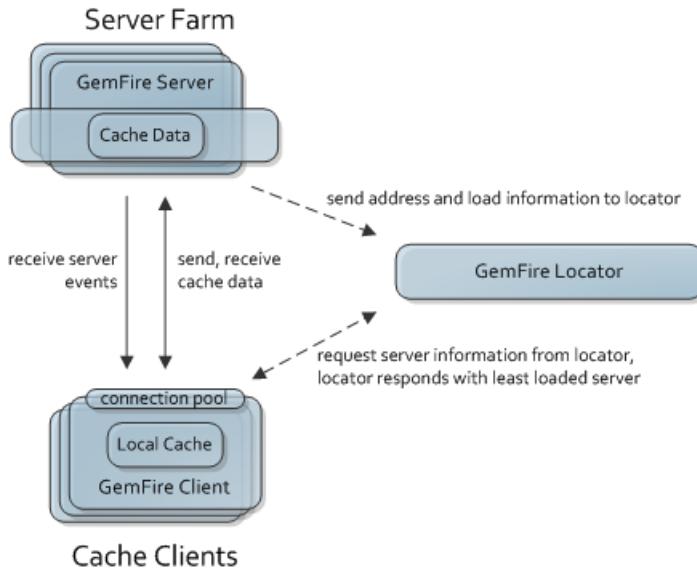
In the client/server architecture, a relatively small server farm manages the cached data of and access to the same data for many client applications. Clients can update and access data efficiently, leaving the servers to manage data distribution to other clients and any synchronization with outside data stores.

### Standard Client/Server Deployment

In the most common client/server topology, a farm of cache servers provides caching services to many clients. Cache servers have a homogeneous data store in data regions that are replicated or partitioned across the server farm.

The client/server data flow proceeds as follows:

- Cache servers send their address and load information to the server locator, if locators are used.
- If locators are used, clients request server connection information from the locator. The locator responds with the address of the least-loaded server.
- The client pool checks its connections periodically for proper server load balancing. The pool rebalances as needed.
- Clients can subscribe to events at startup. Events are streamed automatically from the servers to client listeners and into the client cache.
- Client data updates and data requests that the client cache does not fulfill are forwarded automatically to the servers.



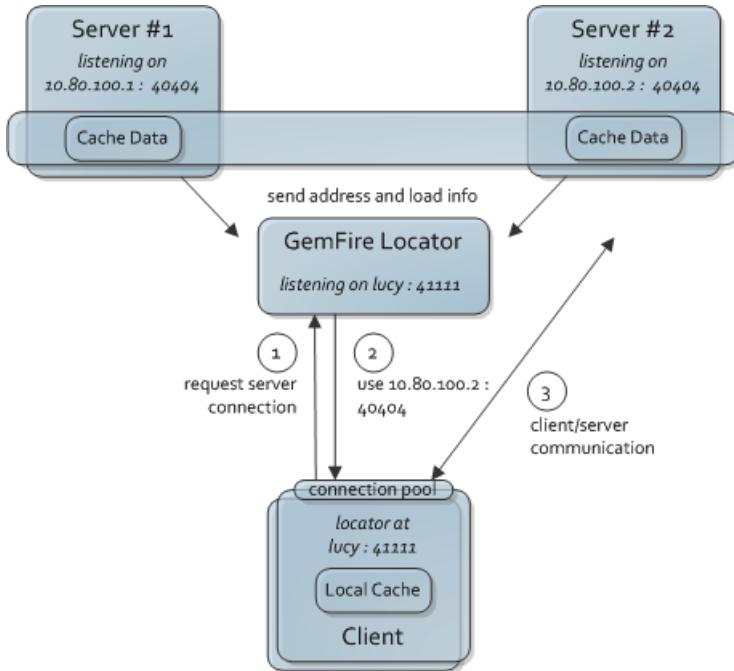
## How Server Discovery Works

vFabric GemFire locators provide reliable and flexible server discovery services for your clients. You can use all servers for all client requests, or group servers according to function, with the locators directing each client request to the right group of servers.

By default, VMware® vFabric™ GemFire® clients and servers discover each other on a predefined port (40404) on the localhost. This works, but is not typically the way you would deploy a client/server configuration. The recommended solution is to use one or more dedicated locators. A locator provides both discovery and load balancing services. With server locators, clients are configured with a locator list and locators maintain a dynamic server list. The locator listens at an address and port for connecting clients and gives the clients server information. The clients are configured with locator information and have no configuration specific to the servers.

### Basic Configuration

In this figure, only one locator is shown, but the recommended configuration uses multiple locators for high availability.



The locator and servers have the same peer discovery configured in their `gemfire.properties`:

```
locators=lucy[41111]
```

The servers, run on their respective hosts, have this `cache-server` configuration in their `cache.xml`:

```
<cache-server port="40404" ...
```

The client's `cache.xml` pool configuration and `region-attributes`:

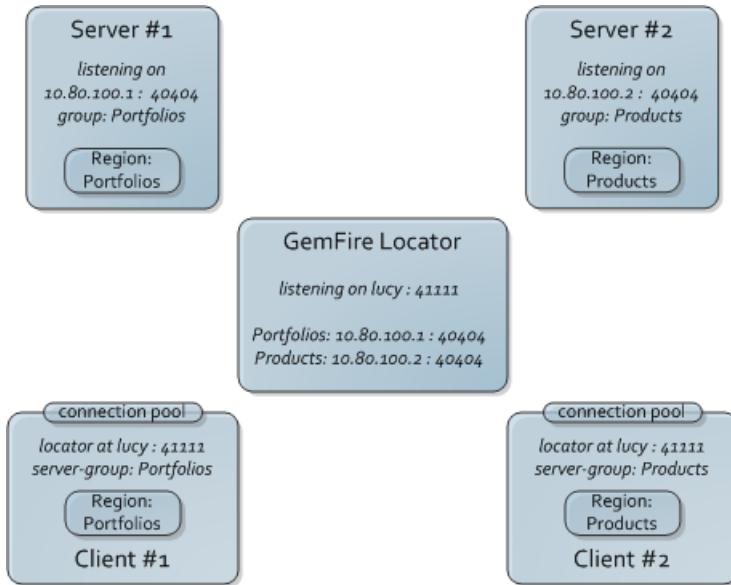
```
<pool name="PoolA" ...
      <locator host="lucy" port="41111">
      <region ...
      <region-attributes pool-name="PoolA" ...
```

## Using Member Groups

You can control which servers are used with named member groups. Do this if you want your servers to manage different data sets or to direct specific client traffic to a subset of servers, such as those directly connected to a back-end database.

To split data management between servers, configure some servers to host one set of data regions and some to host another set. Assign the servers to two separate member groups. Then, define two separate server pools on the client side and assign the pools to the proper corresponding client regions.

In this figure, the client use of the regions is also split, but you could have both pools and both regions defined in all of your clients.



This is the `gemfire.properties` definition for Server 1:

```
#gemfire.properties
groups=Portfolios
```

And the pool declaration for Client 1:

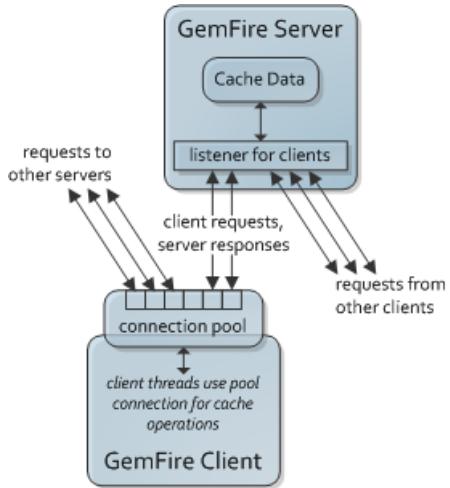
```
<pool name="PortfolioPool" server-group="Portfolios"...
    <locator host="lucy" port="41111">
```

## How Client/Server Connections Work

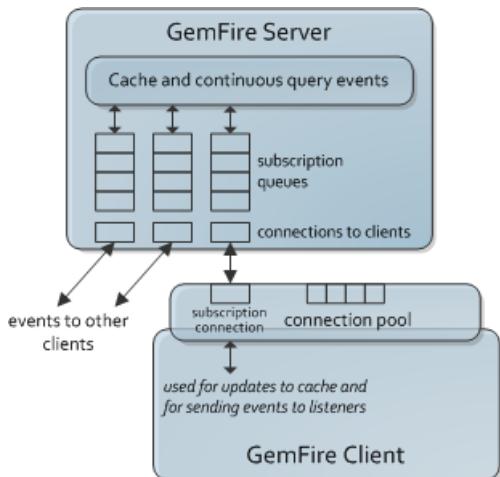
The server pools in your vFabric GemFire client processes manage all client connection requests to the server tier. To make the best use of the pool functionality, you should understand how the pool manages the server connections.

Client/server communication is done in two distinct ways. Each kind of communication uses a different type of connection for maximum performance and availability.

- **Pool connections.** The pool connection is used to send individual operations to the server to update cached data, to satisfy a local cache miss, or to run an ad hoc query. Each pool connection goes to a host/port location where a server is listening. The server responds to the request on the same connection. Generally, client threads use a pool connection for an individual operation and then return the connection to the pool for reuse, but you can configure to have connections owned by threads. This figure shows pool connections for one client and one server. At any time, a pool may have from zero to many pool connections to any of the servers.



- **Subscription connections.** The subscription connection is used to stream cache and continuous query events from the server to the client. To use this, set the client attribute `subscription-enabled` to true. The server establishes a queue to asynchronously send subscription events and the pool establishes a subscription connection to handle the incoming messages. The events sent depend on how the client subscribes.



## How the Pool Chooses a Server Connection

The pool gets server connection information from the server locators or, alternately, from the static server list.

- **Server Locators.** Server locators maintain information about which servers are available and which has the least load. New connections are sent to the least loaded servers. The pool requests server information from a locator when it needs a new connection. The pool randomly chooses the locator to use and the pool sticks with a locator until the connection fails.
- **Static Server List.** If you use a static server list, the pool shuffles it once at startup, to provide randomness between clients with the same list configuration, and then runs through the list round robin connecting as needed to the next server in the list. There is no load balancing or dynamic server discovery with the static server list.

## How the Pool Connects to a Server

When a pool needs a new connection, it goes through these steps until either it successfully establishes a connection, it has exhausted all available servers, or the `free-connection-timeout` is reached.

1. Requests server connection information from the locator or retrieves the next server from the static server list.
2. Sends a connection request to the server.

If the pool fails to connect while creating a subscription connection or provisioning the pool to reach the `min-connections` setting, it logs a fine level message and retries after the time indicated by `ping-interval`.

If an application thread calls an operation that needs a connection and the pool can't create it, the operation returns a `NoAvailableServersException`.

## How the Pool Manages Pool Connections

Each `Pool` instance in your client maintains its own connection pool. The pool responds as efficiently as possible to connection loss and requests for new connections, opening new connections as needed. When you use a pool with the server locator, the pool can quickly respond to changes in server availability, adding new servers and disconnecting from unhealthy or dead servers with little or no impact on your client threads. Static server lists require more close attention as the client pool is only able to connect to servers at the locations specified in the list.

The pool adds a new pool connection when one of the following happens:

- The number of open connections is less than the `Pool`'s `min-connections` setting.
- A thread needs a connection, all open connections are in use, and adding another connection would not take the open connection count over the pool's `max-connections` setting. If the `max-connections` setting has been reached, the thread blocks until a connection becomes available.

The pool closes a pool connection when one of the following occurs:

- The client receives a connectivity exception from the server.
- The server doesn't respond to a direct request or ping within the client's configured `read-timeout` period. In this case, the pool removes all connections to that server.
- The number of pool connections exceeds the pool's `min-connections` setting and the client doesn't send any requests over the connection for the `idle-timeout` period.

When it closes a connection that a thread is using, the pool switches the thread to another server connection, opening a new one if needed.

## How the Pool Manages Subscription Connections

The pool's subscription connection is established in the same way as the pool connections, by requesting server information from the locator and then sending a request to the server, or, if you are using a static server list, by connecting to the next server in the list.

Subscription connections remain open for as long as needed and are not subject to the timeouts that apply to pool connections.

## How the Pool Conditions Server Load

When locators are used, the pool periodically conditions its pool connections. Each connection has an internal lifetime counter. When the counter reaches the configured `load-conditioning-interval`, the pool checks with the locator to see if the connection is using the least loaded server. If not, the pool establishes a new connection to the least loaded server, silently puts it in place of the old connection, and closes the old connection. In either case, when the operation completes, the counter starts at zero. Conditioning happens behind the scenes

and does not affect your application's connection use. This automatic conditioning allows very efficient upscaling of your server pool. It is also useful following planned and unplanned server outages, during which time the entire client load will have been placed on a subset of the normal set of servers.

## Configuring a Client/Server System

Configure your server and client processes and data regions to run your client/server system.

### Prerequisites

- Configure your server system using locators for member discovery. See [Configuring Peer-to-Peer Discovery](#) on page 173 and [Managing a Peer or Server Cache](#) on page 139
- Configure your clients as standalone applications. See [Managing a Client Cache](#) on page 139
- Be familiar with cache region configuration. See [Data Regions](#) on page 145
- Be familiar with server and client configuration properties. See [cache.xml Quick Reference](#) on page 861.

### Procedure

- Configure servers to listen for clients by completing one or both of the following tasks.

- Configure each application server as a server by specifying the `<cache-server>` element in the application's `cache.xml` and optionally specifying a non-default port to listen on for client connections.

For example:

```
<cache-server port="40404" ... />
```

- Optional. Configure each `cacheserver` process with a non-default port to listen on for client connections.

For example:

```
prompt> cacheserver start -port="44454"
```

- Configure clients to connect to servers. In the client `cache.xml`, use the server system's locator list to configure your client server pools and configure your client regions to use the pools.

For example:

```
<client-cache>
  <pool name="publisher" subscription-enabled="true">
    <locator host="lucy" port="41111"/>
    <locator host="lucy" port="41111"/>
  </pool>
  ...
  <region name="clientRegion" ...
    <region-attributes pool-name="publisher" ...
```

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 provide the information to the clients as needed.

- Configure the server data regions for client/server work, following these guidelines. These do not need to be performed in this order.
  - Configure your server regions as partitioned or replicated, to provide a coherent cache view of server data to all clients.



**Note:** If you do not configure your server regions as partitioned or replicated, you can get unexpected results with calls that check server region contents, such as `keySetOnServer` and `containsKeyOnServer`. You might get only partial results, and you might also get inconsistent responses from two consecutive calls. These results occur because the servers report only on their

local cache content and, without partitioned or replicated regions, they might not have a complete view of the data in their local caches.

- b) When you define your replicated server regions, use any of the REPLICATE RegionShortcut settings except for REPLICATE\_PROXY. Replicated server regions must have distributed-ack or global scope, and every server that defines the region must store data. The region shortcuts use distributed-ack scope and all non-proxy settings store data.
- c) When you define your partitioned server regions, use the PARTITION RegionShortcut options. You can have local data storage in some servers and no local storage in others.

When you start the server and client systems, the client regions will use the server regions for cache misses, event subscriptions, querying, and other caching activities.

### What to do next

Configure your clients to use the cache and to subscribe to events from the servers as needed by your application. See [Configuring Client/Server Event Messaging](#) on page 297.

## Organizing Servers Into Logical Member Groups

In a client/server configuration, by putting servers into logical member groups, you can control which servers your clients use and target specific servers for specific data or tasks. You can configure servers to manage different data sets or to direct specific client traffic to a subset of servers, such as those directly connected to a back-end database.

You can also define member groups to deploy JARs in parallel or to perform administrative commands across a member group.

To add servers to a member group, you can configure the following:

1. Add the member group names to the `gemfire.properties` file for the server. For example:

```
#gemfire.properties  
groups=Portfolios,ManagementGroup1
```

A server can belong to more than one member group. If specifying multiple group membership for the server, use a comma-separated list. Alternatively, if you are using the `gfsh` command interface to start up the server, provide a group name as a parameter:

```
gfsh>start server --name=server1 \  
--group=Portfolios,ManagementGroup1
```

2. To configure a client to connect to a specific member group, modify the client's `cache.xml` file to define a distinct pool for each `server-group` and assign the pools to the corresponding client regions:

```
<pool name="PortfolioPool" server-group="Portfolios" ...  
  <locator host="lucy" port="41111">  
  ...  
</pool>  
  ...  
<region name="clientRegion" ...  
  <region-attributes pool-name="PortfolioPool" ...
```

## Client/Server Example Configurations

For easy configuration, you can start with these example client/server configurations and modify for your systems.

## Examples of Standard Client/Server Configuration

Generally, locators and servers use the same distributed system properties file, which lists locators as the discovery mechanism for peer members and for connecting clients. For example:

```
mcast-port=0
locators=localhost[41111]
```

On the machine where you wish to run the locator (in this example, 'localhost'), you can start the locator from a gfsh prompt:

```
gfsh>start locator --name=locator_name --port=41111
```

Or directly from a command line:

```
prompt# gfsh start locator --name=locator_name --port=41111
```

Specify a name for the locator that you wish to start on the localhost.

The server's cache.xml declares a cache-server element, which identifies the JVM as a server in the distributed system.

```
<cache>
  <cache-server port="40404" ... />
  <region ... .
```

Once the locator and server are started, the locator tracks the server as a peer in its distributed system and as a server listening for client connections at port 40404.

The client's cache.xml <client-cache> declaration automatically configures it as a standalone GemFire application.

The client's cache.xml:

- Declares a single connection pool with the locator as the reference for obtaining server connection information.
- Creates cs\_region with the client region shortcut configuration, CACHING\_PROXY. This configures it as a client region that stores data in the client cache.

There is only one pool defined for the client, so the pool is automatically assigned to all client regions.

```
<client-cache>
  <pool name="publisher" subscription-enabled="true">
    <locator host="localhost" port="41111"/>
  </pool>
  <region name="cs_region" refid="CACHING_PROXY">
  </region>
</client-cache>
```

With this, the client is configured to go to the locator for the server connection location. Then any cache miss or put in the client region is automatically forwarded to the server.

### Example: Standalone Publisher Client, Client Pool, and Region

The following API example walks through the creation of a standalone publisher client and the client pool and region.

```
public static ClientCacheFactory connectStandalone(String name) {
    return new ClientCacheFactory()
        .set("log-file", name + ".log")
        .set("statistic-archive-file", name + ".gfs")
        .set("statistic-sampling-enabled", "true")
        .set("cache-xml-file", "")
        .addPoolLocator("localhost", LOCATOR_PORT);
}

private static void runPublisher() {
```

```

ClientCacheFactory ccf = connectStandalone("publisher");
ClientCache cache = ccf.create();
ClientRegionFactory<String, String> regionFactory =
    cache.createClientRegionFactory(PROXY);
Region<String, String> region = regionFactory.create("DATA");

//... do work ...

cache.close();
}

```

### Example: Standalone Subscriber Client

This API example creates a standalone subscriber client using the same `connectStandalone` method as the previous example.

```

private static void runSubscriber() throws InterruptedException {
    ClientCacheFactory ccf = connectStandalone("subscriber");
    ccf.setPoolSubscriptionEnabled(true);
    ClientCache cache = ccf.create();
    ClientRegionFactory<String, String> regionFactory =
        cache.createClientRegionFactory(PROXY);
    Region<String, String> region = regionFactory
        .addCacheListener(new SubscriberListener())
        .create("DATA");
    region.registerInterestRegex(".*", // everything
        InterestResultPolicy.NONE,
        false/*isDurable*/);
    SubscriberListener myListener =
        (SubscriberListener)region.getAttributes().getCacheListeners()[0];
    System.out.println("waiting for publisher to do " + NUM_PUTS + " puts...");

    myListener.waitForPuts(NUM_PUTS);
    System.out.println("done waiting for publisher.");

    cache.close();
}

```

### Example of a Static Server List in Client/Server Configuration

You can specify a static server list instead of a locator list in the client configuration. With this configuration, the client's server information does not change for the life of the client member. You do not get dynamic server discovery, server load conditioning, or the option of logical server grouping. This model is useful for very small deployments, such as test systems, where your server pool is stable. It avoids the administrative overhead of running locators.

This model is also suitable if you must use hardware load balancers. You can put the addresses of the load balancers in your server list and allow the balancers to redirect your client connections.

The client's server specification must match the addresses where the servers are listening.

In the server cache configuration file, here are the pertinent settings.

```

<cache>
  <cache-server port="40404" ... /> <region ...>

```

The client's `cache.xml` file declares a connection pool with the server explicitly listed and names the pool in the attributes for the client region. This XML file uses a region attributes template to initialize the region attributes configuration.

```

<client-cache>
  <pool name="publisher" subscription-enabled="true">

```

```

<server host="localhost" port="40404"/>
</pool>
<region name="cs_region" refid="CACHING_PROXY">
</region>
</client-cache>

```

## Fine-Tuning Your Client/Server Configuration

You can fine-tune your client/server system with server load-balancing and client thread use of pool connections. For example, you can configure how often the servers check their load with the cache server `load-poll-interval` property, or configure your own server load metrics by implementing the `com.gemstone.gemfire.cache.server` package.

### How Server Load Conditioning Works

When the client pool requests connection information from the server locator, the locator returns the least-loaded server for the connection type. The pool uses this "best server" response to open new connections and to condition (rebalance) its existing pool connections.

- The locator tracks server availability and load according to information that the servers provide. Each server probes its load metrics periodically and, when it detects a change, sends new information to the locator. This information consists of current load levels and estimates of how much load would be added for each additional connection. The locator compares the load information from its servers to determine which servers can best handle more connections.
- You can configure how often the servers check their load with the cache server's `load-poll-interval`. You might want to set it lower if you find your server loads fluctuating too much during normal operation. The lower you set it, however, the more overhead your load balancing will use.
- Between updates from the servers, the locators estimate which server is the least loaded by using the server estimates for the cost of additional connections. For example, if the current pool connection load for a server's connections is 0.4 and each additional connection would add 0.1 to its load, the locator can estimate that adding two new pool connections will take the server's pool connection load to 0.6.
- Locators do not share connection information among themselves. These estimates provide rough guidance to the individual locators for the periods between updates from the servers.

GemFire provides a default utility that probes the server and its resource usage to give load information to the locators. The default probe returns the following load metrics:

- The pool connection load is the number of connections to the server divided by the server's `max-connections` setting. This means that servers with a lower `max-connections` setting receives fewer connections than servers with a higher setting. The load is a number between 0 and 1, where 0 means there are no connections, and 1 means the server is at `max-connections`. The load estimate for each additional pool connection is  $1/\text{max-connections}$ .
- The subscription connection load is the number of subscription queues hosted by this server. The load estimate for each additional subscription connection is 1.

To use your own server load metrics instead of the default, implement the `ServerLoadProbe` or `ServerLoadProbeAdapter` and related interfaces and classes in the `com.gemstone.gemfire.cache.server` package. The load for each server is weighed relative to the loads reported by other servers in the system.

### Client Thread Use of Pool Connections

By default, a client thread retrieves a connection from the open connection pool for each forwarded operation and returns the connection to the pool as soon as the request is complete. If your client thread runs a `put` on a client region, for example, that operation grabs a server connection, sends the `put` to the server, and then returns the connection to the pool. This action keeps the connections available to the most threads possible.

### Set Up a Thread-Local (Dedicated) Connection

You configure threads to use dedicated connections by setting `thread-local-connections` to true. In this case the thread holds its connection either until the thread explicitly releases the connection, or the connection expires based on `idle-timeout` or `load-conditioning-interval`.

### Release a Thread-Local Connection

If you use thread-local connections, you should release the connection as soon as your thread finishes its server activities.

- Call `releaseThreadLocalConnection` on the `Pool` instance you are using for the region:

```
Region myRegion ...  
PoolManager.find(myRegion).releaseThreadLocalConnection();
```

# Chapter 26

## Multi-site (WAN) Configuration

Use the multi-site configuration to scale horizontally between disparate, loosely-coupled distributed systems. A wide-area network (WAN) is the main use case for the multi-site topology.

### How Multi-site (WAN) Systems Work

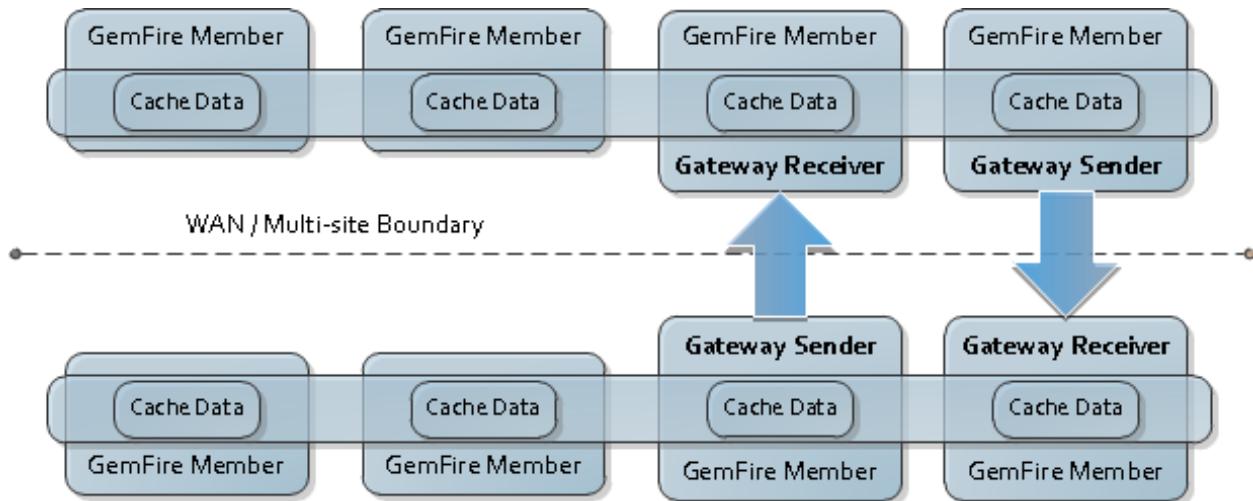
The vFabric GemFire multi-site implementation connects disparate distributed systems. The systems act as one when they are coupled, and they act as independent systems when communication between sites fails. The coupling is tolerant of weak or slow links between distributed system sites. A wide-area network (WAN) is the main use case for the multi-site topology.

### Overview of Multi-site Caching

A multi-site installation consists of two or more distributed systems that are loosely coupled. Each site manages its own distributed system, but region data is distributed to remote sites using one or more logical connections.

The logical connections consist of a gateway sender in the sending site, and a gateway receiver in the receiving site. In a client/server installation, gateway senders and gateway receivers are configured in the server layer.

Gateway senders and receivers are defined at startup in the distributed system member caches. A site can use *serial* and/or *parallel* gateway sender configurations, as described in [Gateway Senders](#) on page 190.



### Consistency for WAN Updates

GemFire ensures that all copies of a region eventually reach a consistent state on all members and clients that host the region, including GemFire members that distribute region events across a WAN.

By default, potential WAN conflicts are resolved using a timestamp mechanism. You can optionally install a custom conflict resolver to apply custom logic when determining whether to apply a potentially conflicting update received over a WAN.

[Consistency for Region Updates](#) on page 239 describes how GemFire ensures consistency within a cluster, in client caches, and when applying updates over a WAN. [Resolving Conflicting Events](#) on page 200 provides more details about implementing a custom conflict resolver for WAN updates.

### Discovery for Multi-Site Systems

Each GemFire cluster in a WAN configuration uses locators to discover remote GemFire clusters as well as local GemFire members.

Each locator in a WAN configuration defines a unique `distributed-system-id` property that identifies the local cluster to which it belongs. A locator uses the `remote-locators` property to define the addresses of one or more locators in remote GemFire clusters to use for WAN distribution.

When a locator starts up, it contacts each locator that is configured in the `remote-locators` property to exchange information about the available locators and gateway receivers in the cluster. The locator also shares information about locators and gateway receivers in any other GemFire clusters that have connected to the cluster. Connected clusters can then use the shared gateway receiver information to distribute region events according to their configured gateway senders.

Each time a new locator starts up or an existing locator shuts down, the changed information is broadcast to other connected GemFire clusters.



**Note:** When you configure a multi-site system, you *must* use locators for discovery rather than multicast.

### Gateway Senders

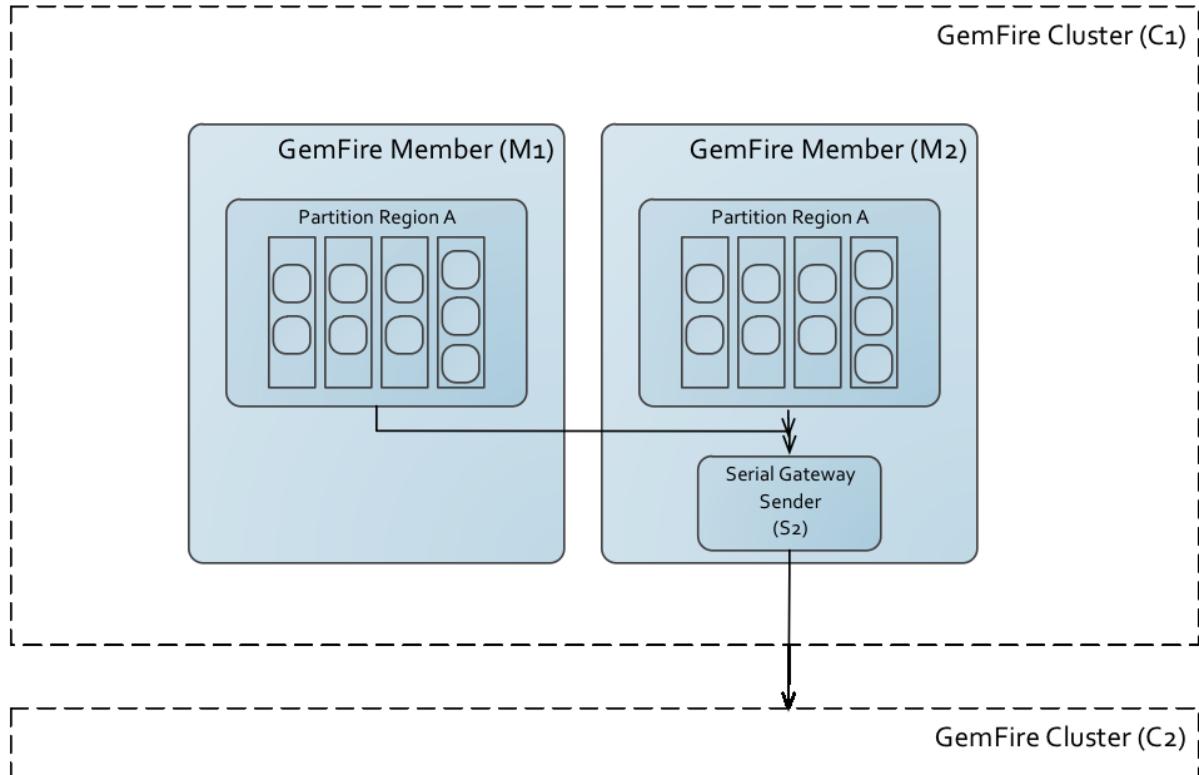
A GemFire cluster uses a *gateway sender* to distribute region events to another, remote GemFire cluster. You can create multiple gateway sender configurations to distribute region events to multiple remote clusters, and/or to distribute region events concurrently to another remote cluster.

A gateway sender always communicates with a gateway receiver in a remote cluster. Gateway senders do not communicate directly with other cache server instances. See [Gateway Receivers](#) on page 193.

GemFire provides two types of gateway sender configurations: *serial* gateway senders and *parallel* gateway senders.

#### Serial Gateway Senders

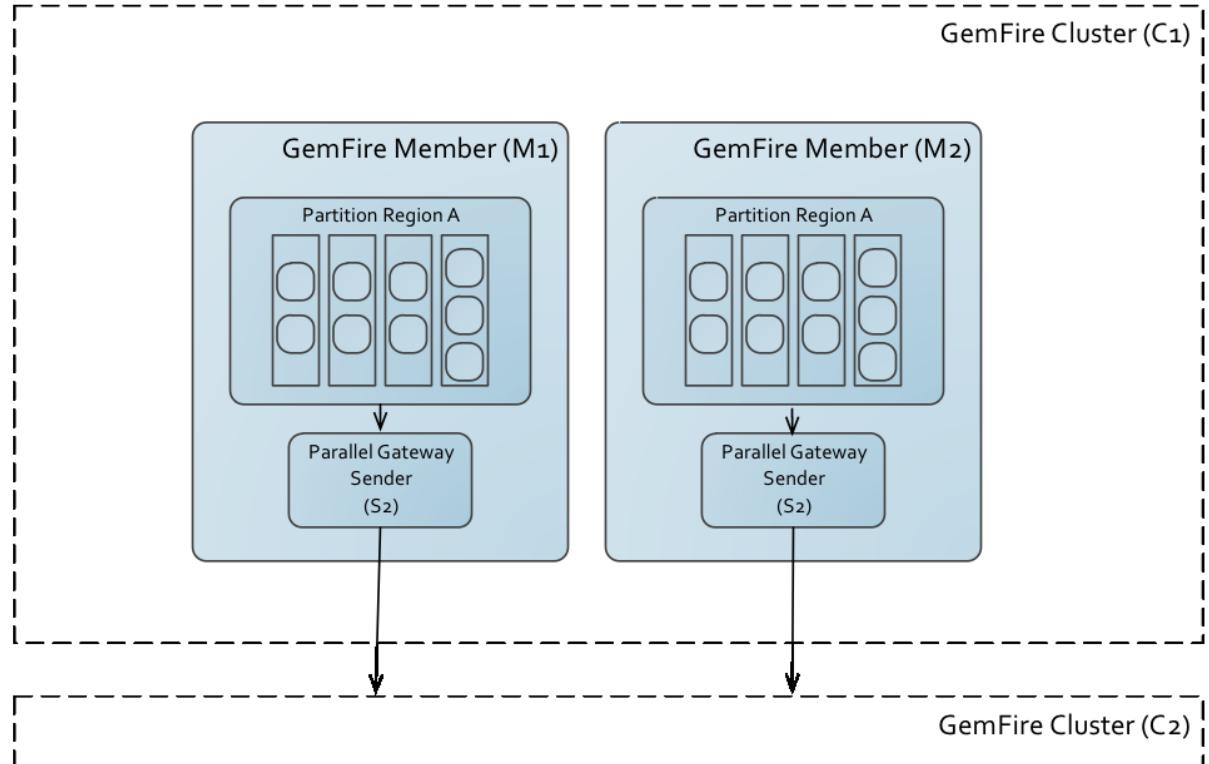
A *serial gateway sender* distributes region events from a single GemFire server in the local cluster to a remote GemFire cluster. Although multiple regions can use the same serial gateway for distribution, a serial gateway uses a single logical event queue to dispatch events for all regions that use the gateway sender.



Because a serial gateway sender distributes all of a region's events from a single GemFire member, it provides the most control over ordering region events as they are distributed across the WAN. However, a serial gateway sender provides only a finite amount of throughput for distributing events. As you add more regions and servers to the local cluster, you may need to configure additional serial gateway senders manually and isolate individual regions on specific serial gateway senders to handle the increased distribution traffic.

### Parallel Gateway Senders

A *parallel gateway sender* distributes region events simultaneously from each GemFire server that hosts the region. For a partitioned region, this means that each GemFire server that hosts buckets for the region uses its own logical queue to distribute events for those buckets. As you add new GemFire servers to scale the partitioned region, WAN distribution throughput scales automatically with each new instance of the parallel gateway sender.



A parallel gateway sender uses multiple queues on multiple GemFire members to simultaneously distribute region events to a remote GemFire cluster. Each queue distributes only part of the events for a configured region. For example, for a partitioned region, each queue distributes only those events for the local partition.

Distributed, non-replicated regions can also use a parallel gateway sender for distribution. With a distributed region, GemFire creates a separate gateway sender and queue on each member that hosts the region, and then hashes region events to place them into a distinct queue. This provides a level of scalability for distributed regions that is similar to that provided for partitioned regions.



**Note:** Replicated regions cannot use a parallel gateway sender.

Although parallel gateway senders provide the best throughput for WAN distribution, they provide less control for ordering events. With a parallel gateway sender, you cannot preserve event ordering for the region as a whole because multiple GemFire servers distribute the regions events at the same time. However, the ordering of events for a given partition (or in a given queue of a distributed region) can be preserved. See [Configuring Event Queues](#) on page 309.

### Gateway Sender Queues

The queue that a gateway sender uses to distribute events to a remote site overflows to disk as needed, in order to prevent the GemFire member from running out of memory. You can configure the maximum amount of memory that each queue uses, as well as the batch size and frequency for processing batches in the queue. You can also configure these queues to persist to disk, so that a gateway sender can pick up where it left off when its member shuts down and is later restarted.

Optionally, a serial gateway sender queue can use multiple threads to dispatch queued events. When you configure multiple threads for a serial gateway sender, the logical queue that is hosted on a GemFire member is divided

into multiple physical queues, each with a dedicated dispatcher thread. You can then configure whether those threads dispatch queued events by key, by thread, or in the same order in which events were added to the queue.



**Note:** You cannot configure multiple dispatcher threads or the ordering policy for a parallel gateway sender queue.

See [Configuring Event Queues](#) on page 309.

## High Availability for Gateway Senders

When a serial gateway sender configuration is deployed to multiple GemFire members, only one "primary" sender is active at a given time. All other serial gateway sender instances are inactive "secondaries" that are available as backups if the primary sender shuts down. GemFire designates the first gateway sender to start up as the primary sender, and all other senders become secondaries. As gateway senders start and shut down in the distributed system, GemFire ensures that the oldest running gateway sender operates as the primary.

A parallel gateway sender is deployed to multiple GemFire members by default, and each GemFire member that hosts primary buckets for a partitioned region actively distributes data to the remote GemFire site. When you use parallel gateway senders, high availability for WAN distribution is provided if you configure the partitioned region for redundancy. With a redundant partitioned region, if a member that hosts primary buckets fails or is shut down, then a GemFire member that hosts a redundant copy of those buckets takes over WAN distribution for those buckets.

## Gateway Receivers

A gateway receiver configures a physical connection for receiving region events from gateway senders in one or more remote GemFire clusters.

A gateway receiver applies each region event to the same region or partition that is hosted in the local GemFire member. (An exception is thrown if the receiver receives an event for a region that it does not define.)

Gateway senders use any available gateway receiver in the target cluster to send region events. You can deploy gateway receiver configurations to multiple GemFire members as needed for high availability and load balancing.

See [Configure Gateway Receivers](#) on page 198.

## Multi-site (WAN) Topologies

To configure your multi-site topology, you should understand the recommended topologies and the topologies to avoid.

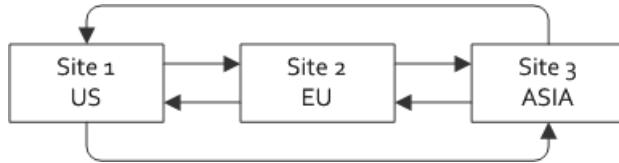
This section describes GemFire's support for various topologies. Depending on your application needs, there may be several topologies that work. These are considerations to keep in mind:

- When a GemFire site receives a message from a gateway sender, it forwards it to the other sites it knows about, excluding those sites that it knows have already seen the message. Each message contains the initial sender's ID and the ID of each of the sites the initial sender sent to, so no site forwards to those sites. However, messages do not pick up the ID of the sites they pass through, so it is possible in certain topologies for more than one copy of a message to be sent to one site.
- In some configurations, the loss of one site affects how other sites communicate with one another.

### Parallel Multi-site Topology

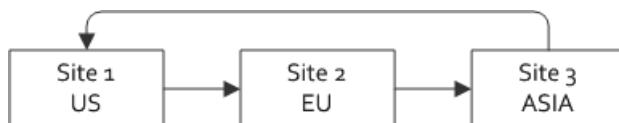
A parallel network topology is one where all sites know about each other. This is a robust configuration, where any one of the sites can go down without disrupting communication between the other sites. A parallel topology also guarantees that no site receives multiple copies of the same message.

The parallel topology for three sites is shown in this figure. In this scenario, if site 1 sends an update to site 2, site 2 forwards to site 3. If site 1 sends an update to sites 2 and 3, neither forwards to the other. Likewise for any other initiating site. If any site is removed, the remaining two maintain a parallel topology.



### Serial Multi-site Topology

A serial network topology is one where each site only knows about one other site. Data is passed from one site to the next serially. This figure shows the topology for three sites. In this scenario, if site 1 sends updates to site 2, site 2 forwards to site 3. Site 3 does not send the updates back to site 1.

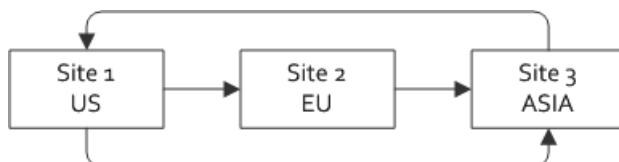


The serial topology guarantees that every site receives one copy of each message sent by any site. With a serial installation, every site must stay up to maintain the topology. The failure of any site breaks the serial link. If site 2 went down, for example, site 3 could send to site 1, but site 1 could not send to site 3.

### Hybrid Multi-site Topology

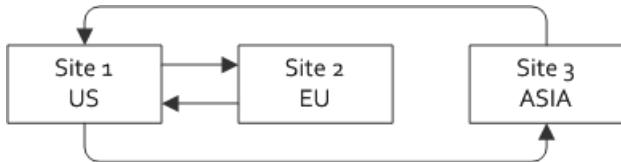
A hybrid network topology contains both parallel and serial connections. There are numerous such configurations.

The following figure shows a hybrid topology composed of a serial topology with an additional link directly from site 1 to site 3, creating a parallel topology between sites 1 and 3. With this topology, no site can receive the same update twice.



With this hybrid topology, if site 2 went down, it would not affect communication between sites 1 and 3 because both could still send to the other. If site 3 went down, however, site 2 would not be able to send to site 1.

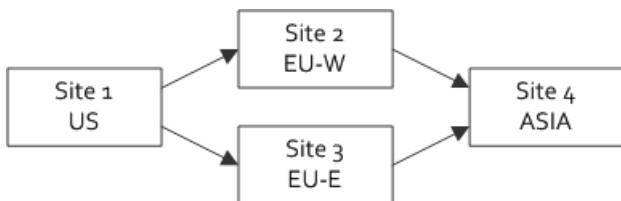
In the figure below, sites 2 and 3 are isolated from one another. This topology might work for an application where there is one producer (site 1) and multiple consumers that have nothing to gain from being connected to each other. This topology also guarantees that no site receives the same update twice.



## Unsupported Topologies

This section lists topologies that do not work and are unsupported.

In the topology shown in the next figure, some sites might receive multiple copies of the same message. If site 1 sends a message to sites 2 and 3, sites 2 and 3 will both forward to site 4.



## Configuring a Multi-site (WAN) System

Plan and configure your multi-site topology, and configure the regions that will be shared between systems.

### Prerequisites

Before you start, you should understand how to configure membership and communication in peer-to-peer systems using locators. See [Configuring Peer-to-Peer Discovery](#) on page 173 and [Configuring Peer Communication](#) on page 174. WAN replication is not supported when using multicast for discovery.

WAN deployments increase the messaging demands on a GemFire system. To avoid hangs related to WAN messaging, always set `conserve-sockets=false` for GemFire members that participate in a WAN deployment. See [Controlling Socket Use](#) on page 520 and [Making Sure You Have Enough Sockets](#) on page 533.



**Note:** If you will be upgrading an existing WAN deployment to use the new WAN API, also refer to the [WAN deployment upgrade guidelines](#) in the vFabric GemFire Release Notes.

### Main Steps

Use the following steps to configure a multi-site system:

1. Plan the topology of your multi-site system. See [Multi-site \(WAN\) Topologies](#) on page 193 for a description of different multi-site topologies.
2. Configure membership and communication for each distributed system in your multi-site system. You must use locators for peer discovery in a WAN configuration. See [Configuring Peer-to-Peer Discovery](#) on page 173. Start each distributed system using a unique `distributed-system-id` and identify remote clusters using `remote-locators`.
3. Configure the gateway senders that you will use to distribute region events to remote systems. See [Configure Gateway Senders](#) on page 196.
4. Create the data regions that you want to participate in the multi-site system, specifying the gateway sender(s) that each region should use for WAN distribution. Configure the same regions in the target clusters to apply the distributed events. See [Create Data Regions for Multi-site Communication](#) on page 197.

5. Configure gateway receivers in each GemFire cluster that will receive region events from another cluster. See [Configure Gateway Receivers](#) on page 198.
6. Start distributed system member processes in the correct order (locators first, followed by data nodes) to ensure efficient discovery of WAN resources. See [Starting Up Your System](#) on page 637.
7. (Optional.) Deploy custom conflict resolvers to handle resolve potential conflicts that are detected when applying events from over a WAN. See [Resolving Conflicting Events](#) on page 200.
8. (Optional.) Deploy WAN filters to determine which events are distributed over the WAN, or to modify events as they are distributed over the WAN. See [Filtering Events for Multi-Site \(WAN\) Distribution](#) on page 199.
9. (Optional.) Configure persistence, conflation, and/or dispatcher threads for gateway sender queues using the instructions in [Configuring Event Queues](#) on page 309.

## Configure Gateway Senders

Each gateway sender configuration

- A unique ID for the gateway sender configuration.
- The distributed system ID of the remote site to which the sender propagates region events.
- A property that specifies whether the gateway sender is a serial gateway sender or a parallel gateway sender.
- Optional properties that configure the gateway sender queue. These queue properties determine features such as the amount of memory used by the queue, whether the queue is persisted to disk, and how one or more gateway sender processes dispatch events from the queue.

See [List item](#) on page 862 for more information about individual configuration properties.

1. For each GemFire system, choose the members that will host a gateway sender configuration and distribute region events to remote sites:
  - You must deploy a parallel gateway sender configuration on each GemFire member that hosts a region that uses the sender.
  - You may choose to deploy a serial gateway sender configuration on one or more GemFire members in order to provide high availability. However, only one instance of a given serial gateway sender configuration distributes region events at any given time.
2. Configure each gateway sender on a GemFire member using either the `cache.xml` file or Java API:

### • cache.xml configuration

These example `cache.xml` entries configure two parallel gateway senders to distribute region events to two remote GemFire clusters (clusters "2" and "3"):

```
<cache>
  <gateway-sender id="sender2" parallel="true"
    remote-distributed-system-id="2"/>
  <gateway-sender id="sender3" parallel="false"
    remote-distributed-system-id="3"/>
  ...
</cache>
```

### • Java configuration

This example code shows how to configure a parallel gateway sender using the GemFire API:

```
// Create or obtain the GemFire cache
Cache cache = new CacheFactory().create();

// Configure and create the gateway sender
GatewaySenderFactory gateway = cache.createGatewaySenderFactory();
gateway.setParallel(true);
GatewaySender sender = gateway.create("sender2", "2");
sender.start();
```

3. Depending on your applications, you may need to configure additional features in each gateway sender.

Things you need to consider are:

- The maximum amount of memory each gateway sender queue can use. When the queue exceeds the configured amount of memory, the contents of the queue overflow to disk. For example:

```
<gateway-sender id="sender2" parallel="true"
  remote-distributed-system-id="2"
  maximum-queue-memory="150"/>
```

- Whether to enable disk persistence, and whether to use a named disk store for persistence or for overflowing queue events. See [Persisting an Event Queue](#) on page 309. For example:

```
<gateway-sender id="sender2" parallel="true"
  remote-distributed-system-id="2"
  is-persistent="true" disk-store-name="cluster2Store"
  maximum-queue-memory="150"/>
```

- (For serial gateway senders only.) Whether to use multiple dispatcher threads on each gateway queue to process the queue's events. The `dispatcher-threads` attribute of the gateway sender determines whether multiple threads process the queue. If you use multiple dispatcher threads, also configure the ordering policy to use for dispatching the events. For example:

```
<gateway-sender id="sender2" parallel="false"
  remote-distributed-system-id="2"
  dispatcher-threads="2" order-policy="partition"/>
```

You cannot configure multiple dispatcher threads or an ordering policy for a parallel gateway sender queue.



**Note:** When you configure multiple dispatcher threads, each thread operates on its own copy of the gateway sender queue. Queue configuration attributes such as `maximum-queue-memory` are repeated for each dispatcher thread that you configure.

See [Configuring Dispatcher Threads and Order Policy for Event Distribution](#) on page 311.

- Determine whether you should conflate events in the queue. See [Conflating Events in a Queue](#) on page 314.



**Note:** The gateway sender configuration for a specific sender `id` must be identical on each GemFire member that hosts the gateway sender.

## Create Data Regions for Multi-site Communication

After you define gateway senders, configure regions to use the gateway sender to distribute region events :

- cache.xml Configuration**

Use the `gateway-sender-ids` region attribute to add gateway senders to a region. To assign multiple gateway senders, use a comma-separated list. For example:

```
<region-attributes gateway-sender-ids="sender2, sender3">
</region-attributes>
```

- Java API Configuration**

This example shows adding two gateway senders (configured in the earlier example) to a partitioned region:

```
RegionFactory rf =
  cache.createRegionFactory(RegionShortcut.PARTITION);
rf.addCacheListener(new LoggingCacheListener());
rf.addGatewaySenderId("sender2");
rf.addGatewaySenderId("sender3");
custRegion = rf.create("customer");
```



**Note:** When using the Java API, you must configure a parallel gateway sender *before* you add its id to a region. This ensures that the sender distributes region events that were persisted before new cache operations take place. If the gateway sender id does not exist when you add it to a region, you receive an `IllegalStateException`.



**Note:** Replicated regions cannot use a parallel gateway sender. Use a serial gateway sender instead.



**Note:** In addition to configuring regions with gateway senders to distribute events, you must configure the same regions in the target clusters to apply the distributed events. The region name in the receiving cluster must exactly match the region name in the sending cluster.

## Configure Gateway Receivers

Always configure a gateway receiver in each GemFire cluster that will receive and apply region events from another cluster.

A gateway receiver configuration can be applied to multiple GemFire servers for load balancing and high availability. However, each GemFire member that hosts a gateway receiver must also define all of the regions for which the receiver may receive an event. If a gateway receiver receives an event for a region that the local member does not define, GemFire throws an exception. See [Create Data Regions for Multi-site Communication](#) on page 197.

A gateway receiver configuration specifies a range of possible port numbers on which to listen. The GemFire server picks an unused port number from the specified range to use for the receiver process. You can use this functionality to easily deploy the same gateway receiver configuration to multiple members:

- **cache.xml Configuration**

The following configuration defines a gateway receiver that listens on an unused port in the range from 1530 to 1550:

```
<cache>
  <gateway-receiver start-port="1530" end-port="1551"/>
  ...
</cache>
```

- **Java Configuration**

```
// Create or obtain the GemFire cache
Cache cache = new CacheFactory().create();

// Configure and create the gateway receiver
GatewayReceiverFactory gateway = cache.createGatewayReceiverFactory();
gateway.setStartPort(1530);
gateway.setEndPort(1551);
GatewayReceiver receiver = gateway.create();
receiver.start();
```



**Note:** When using the Java API, you must create any region that might receive events from a remote site before you start the gateway receiver. Otherwise, batches of events could arrive from remote sites before the regions for those events have been created. If this occurs, the local site will throw exceptions because the receiving region does not exist yet. If you define regions in `cache.xml`, the correct startup order is handled automatically.

## Filtering Events for Multi-Site (WAN) Distribution

You can optionally create gateway sender and/or gateway receiver filters to control which events are queued and distributed to a remote site, or to modify the data stream that is transmitted between GemFire sites.

You can implement and deploy two different types of filter for multi-site events:

- **GatewayEventFilter**. A `GatewayEventFilter` implementation determines whether a region event is placed in a gateway sender queue and/or whether an event in a gateway queue is distributed to a remote site. You can optionally add one or more `GatewayEventFilter` implementations to a gateway sender, either in the `cache.xml` configuration file or using the Java API.

GemFire makes a synchronous call to the filter's `beforeEnqueue` method before it places a region event in the gateway sender queue. The filter returns a boolean value that specifies whether the event should be added to the queue.

GemFire asynchronously calls the filter's `beforeTransmit` method to determine whether the gateway sender dispatcher thread should distribute the event to a remote gateway receiver.

For events that are distributed to another site, GemFire calls the listener's `afterAcknowledgement` method to indicate that it has received an ack from the remote site after the event was received.

- **GatewayTransportFilter**. Use a `GatewayTransportFilter` implementation to process the TCP stream that sends a batch of events that is distributed from one GemFire cluster to another over a WAN. A `GatewayTransportFilter` is typically used to perform encryption or compression on the data that is distributed. You install the same `GatewayTransportFilter` implementation on both a gateway sender and gateway receiver.

When a gateway sender processes a batch of events for distribution, GemFire delivers the stream to the `getInputStream` method of a configured `GatewayTransportFilter` implementation. The filter processes and returns the stream, which is then transmitted to the gateway receiver. When the gateway receiver receives the batch, GemFire calls the `getOutputStream` method of a configured filter, which again processes and returns the stream so that the events can be applied in the local cluster.

## Configuring Multi-Site Event Filters

You install a `GatewayEventFilter` implementation to a configured gateway sender in order to decide which events are queued and distributed. You install a `GatewayTransportFilter` implementation to both a gateway sender and a gateway receiver to process the stream of batched events that are distributed between two sites:

### • XML example

```
<cache>
  <gateway-sender id="remoteA" parallel="true"
  remote-distributed-system-id="1">
    <gateway-event-filter>
      <class-name>com.gemstone.gemfire.util.SampleEventFilter</class-name>

      <parameter name="param1">
        <string>"value1"</string>
      </parameter>
    </gateway-event-filter>
    <gateway-transport-filter>

      <class-name>com.gemstone.gemfire.util.SampleTransportFilter</class-name>
      <parameter name="param1">
        <string>"value1"</string>
      </parameter>
    </gateway-transport-filter>
```

```

        </gateway-sender>
    </cache>

<cache>
    ...
    <gateway-receiver start-port="1530" end-port="1551">
        <gateway-transport-filter>

<class-name>com.gemstone.gemfire.util.SampleTransportFilter</class-name>
            <parameter name="param1">
                <string>"value1"</string>
            </parameter>
        </gateway-transport-filter>
    </gateway-receiver>
</cache>
```

- **API example**

```

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

GatewayEventFilter efilter = new SampleEventFilter();
GatewayTransportFilter tfilter = new SampleTransportFilter();

GatewaySenderFactory gateway = cache.createGatewaySenderFactory();
gateway.setParallel(true);
gateway.addGatewayEventFilter(efilter);
gateway.addTransportFilter(tfilter);
GatewaySender sender = gateway.create("remoteA", "1");
sender.start();

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

GatewayTransportFilter tfilter = new SampleTransportFilter();

GatewayReceiverFactory gateway = cache.createGatewayReceiverFactory();
gateway.setStartPort(1530);
gateway.setEndPort(1551);
gateway.addTransportFilter(tfilter);
GatewayReceiver receiver = gateway.create();
receiver.start();
```

## Resolving Conflicting Events

You can optionally create a `GatewayConflictResolver` cache plug-in to decide whether a potentially conflicting event that was delivered from another site should be applied to the local cache.

By default, all regions perform consistency checks when a member applies an update received either from another cluster member or from a remote cluster over the WAN. The default consistency checking for WAN events is described in [How Consistency Is Achieved in WAN Deployments](#) on page 244.

You can override the default consistency checking behavior by writing and configuring a custom `GatewayConflictResolver`. The `GatewayConflictResolver` implementation can use the timestamp and distributed system ID included in a WAN update event to determine whether or not to apply the update. For example, you may decide that updates from a particular cluster should always "win" a conflict when the timestamp difference between updates is less than some fixed period of time.

Pivotal GemFire installs an example `GatewayConflictResolver` implementation in the `SampleCode\examples\dist\wanActiveActive` subdirectory of the GemFire installation.

## Implementing a GatewayConflictResolver



**Note:** A `GatewayConflictResolver` implementation is called only for update events that could cause a conflict in the region. This corresponds to update events that have a different distributed system ID than the distributed system that last updated the region entry. If the same distributed system ID makes consecutive updates to a region entry, no conflict is possible, and the `GatewayConflictResolver` is not called.

### Procedure

1. Program the event handler:
  - a. Create a class that implements the `GatewayConflictResolver` interface. See the example conflict resolver installed in the `SampleCode\examples\dist\wanActiveActive` subdirectory of the GemFire installation.
  - b. If you want to declare the handler in `cache.xml`, implement the `com.gemstone.gemfire.cache.Declarable` interface as well.
  - c. Implement the handler's `onEvent()` method to determine whether the WAN event should be allowed. `onEvent()` receives both a `TimestampedEntryEvent` and a `GatewayConflictHelper` instance. `TimestampedEntryEvent` has methods for obtaining the timestamp and distributed system ID of both the update event and the current region entry. Use methods in the `GatewayConflictHelper` to either disallow the update event (retaining the existing region entry value) or provide an alternate value.

#### Example:

```
public void onEvent(TimestampedEntryEvent event, GatewayConflictHelper
    helper) {
    if (event.getOperation().isUpdate()) {
        ShoppingCart oldCart = (ShoppingCart)event.getOldValue();
        ShoppingCart newCart = (ShoppingCart)event.getNewValue();
        oldCart.updateFromConflictingState(newCart);
        helper.changeEventValue(oldCart);
    }
}
```



**Note:** In order to maintain consistency in the region, your conflict resolver must always resolve two events in the same way regardless of which event it receives first.

2. Install the conflict resolver for the cache, using either the `cache.xml` file or the Java API.

#### cache.xml

```
<cache>
  ...
  <gateway-conflict-resolver>
    <class-name>myPackage.MyConflictResolver</class-name>
  </gateway-conflict-resolver>
  ...
</cache>
```

#### Java API

```
// Create or obtain the GemFire cache
Cache cache = new CacheFactory().create();

// Create and add a conflict resolver
cache.setGatewayConflictResolver(new MyConflictResolver);
```



# Part 6

---

## Developing with vFabric GemFire

*Developing with vFabric GemFire* explains main concepts of application programming with vFabric GemFire. It describes how to plan and implement regions, data serialization, event handling, delta propagation, transactions, and more.



**Note:** Although this section primarily discusses using the GemFire cache.xml file to configure GemFire and using the GemFire API to write applications, Java developers can also use the Spring Data GemFire API to configure a distributed GemFire system and write applications that access and use data stored in GemFire. Spring Data GemFire provides many advantages for creating such applications with GemFire.

For more information on the advantages of using Spring with GemFire, see [Getting Started with Spring Data GemFire](#) on page 113.

For a complete discussion of Spring Data GemFire, see [Spring Data GemFire Reference Guide](#)

### Topics:

- [Region Data Storage and Distribution](#)
- [Partitioned Regions](#)
- [Distributed and Replicated Regions](#)
- [Consistency for Region Updates](#)
- [General Region Data Management](#)
- [Data Serialization](#)
- [Events and Event Handling](#)
- [Delta Propagation](#)
- [Querying](#)
- [Continuous Querying](#)
- [Transactions](#)
- [Function Execution](#)



# Chapter 27

---

## Region Data Storage and Distribution

The vFabric GemFire data storage and distribution models put your data in the right place at the right time. You should understand all the options for data storage in GemFire before you start configuring your data regions.

### Storage and Distribution Options

GemFire provides several models for data storage and distribution including partitioned or replicated as well as distributed or non-distributed (local cache storage).

At its most general, data management means having current data available when and where your applications need it. In a properly configured VMware® vFabric™ GemFire® installation, you store your data in your local members and GemFire automatically distributes it to the other members that need it according to your cache configuration settings. You may be storing very large data objects that require special consideration, or you may have a high volume of data requiring careful configuration to safeguard your application's performance or memory use. You may need to be able to explicitly lock some data during particular operations. Most data management features are available as configuration options, which you can specify either through the `cache.xml` file or the API. Once configured, GemFire manages the data automatically. For example, this is how you manage data distribution, disk storage, data expiration activities, and data partitioning. A few features are managed at run-time through the API.

At the architectural level, data distribution runs between peers in a single system, between clients and servers, and between separate distributed system sites.

- Peer-to-peer provides the core distribution and storage models, which are specified as attributes on the data regions.
- For client/server, you choose which data regions to share between the client and server tiers. Then, within each region, you can fine-tune the data that the server automatically sends to the client by subscribing to subsets.
- For multi-site, you choose which data regions to share between sites. Because of the high cost of distributing data between disparate geographical locations, not all changes are passed between sites.

Data storage in any type of installation is based on the peer-to-peer configuration for each individual distributed system. Data and event distribution is based on a combination of the peer-to-peer and system-to-system configurations.

### Peer-to-Peer Region Storage and Distribution

The storage and distribution models are configured through cache and region attributes. The main choices are partitioned, replicated, or just distributed. All server regions must be partitioned or replicated. Each region's `data-policy` and `subscription-attributes`, and its `scope` if it is not a partitioned region, interact for finer control of data distribution.

## Storing Data in the Local Cache

To store data in your local cache, use a region `ref id` with a `RegionShortcut` or `ClientRegionShortcut` that has local state. These automatically set the region `data-policy` to a non-empty policy. The regions without storage can send and receive event distributions without storing anything in your application heap. With the other settings, all entry operations received are stored locally.

## Region Types

Region types define region behavior within a single distributed system. You have various options for region data storage and distribution.

Within a vFabric GemFire distributed system, you can define distributed regions and non-distributed regions, and you can define regions whose data is spread across the distributed system, and regions whose data is entirely contained in a single member.

Your choice of region type is governed in part by the type of application you are running. In particular, you need to use specific region types for your servers and clients for effective communication between the two tiers:

- Server regions are created inside a `Cache` by servers and are accessed by clients that connect to the servers from outside the server's distributed system. Server regions must have region type `partitioned` or `replicated`. Server region configuration uses the `RegionShortcut` enum settings.
- Client regions are created inside a `ClientCache` by clients and are configured to distribute data and events between the client and the server tier. Client regions must have region type `local`. Client region configuration uses the `ClientRegionShortcut` enum settings.
- Peer regions are created inside a `Cache`. Peer regions may be server regions, or they may be regions that are not accessed by clients. Peer regions can have any region type. Peer region configuration uses the `RegionShortcut` enum settings.



### Note:

When you configure a server or peer region using the `cache.xml` file, you can use *region shortcuts* to define the basic configuration of your region. A region shortcut provides a set of default configuration attributes that are designed for various types of caching architectures. You can then add additional configuration attributes as needed to customize your application. For more information and a complete reference of these region shortcuts, see [Region Shortcuts Reference](#) on page 963.

These are the primary configuration choices for each data region.

Region Type	Description	Best suited for...
Partitioned	System-wide setting for the data set. Data is divided into buckets across the members that define the region. For high availability, configure redundant copies so each bucket is stored in multiple members with one member holding the primary.	Server regions and peer regions <ul style="list-style-type: none"> <li>• Very large data sets</li> <li>• High availability</li> <li>• Write performance</li> <li>• Partitioned event listeners and data loaders</li> </ul>
Replicated (distributed)	Holds all data from the distributed region. The data from the distributed region is copied into the member replica region. Can be mixed with non-replication, with some members holding replicas and some holding non-replicas.	Server regions and peer regions <ul style="list-style-type: none"> <li>• Read heavy, small datasets</li> <li>• Asynchronous distribution</li> <li>• Query performance</li> </ul>
Distributed non-replicated	Data is spread across the members that define the region. Each member holds only the data it has expressed interest in. Can be mixed with replication,	Peer regions, but not server regions and not client regions <ul style="list-style-type: none"> <li>• Asynchronous distribution</li> </ul>

Region Type	Description	Best suited for...
	with some members holding replicas and some holding non-replicas.	<ul style="list-style-type: none"> <li>Query performance</li> </ul>
Non-distributed (local)	The region is visible only to the defining member.	<p>Client regions and peer regions</p> <ul style="list-style-type: none"> <li>Data that is not shared between applications</li> </ul>

## Partitioned Regions

Partitioning is a good choice for very large server regions. Partitioned regions are ideal for data sets in the hundreds of gigabytes and beyond.



**Note:** Partitioned regions generally require more JDBC connections than other region types because each member that hosts data must have a connection.

Partitioned regions group your data into buckets, each of which is stored on a subset of all of the system members. Data location in the buckets does not affect the logical view - all members see the same logical data set.

Use partitioning for:

- **Large data sets.** Store data sets that are too large to fit into a single member, and all members will see the same logical data set. Partitioned regions divide the data into units of storage called buckets that are split across the members hosting the partitioned region data, so no member needs to host all of the region's data. GemFire provides dynamic redundancy recovery and rebalancing of partitioned regions, making them the choice for large-scale data containers. More members in the system can accommodate more uniform balancing of the data across all host members, allowing system throughput (both gets and puts) to scale as new members are added.
- **High availability.** Partitioned regions allow you configure the number of copies of your data that GemFire should make. If a member fails, your data will be available without interruption from the remaining members. Partitioned regions can also be persisted to disk for additional high availability.
- **Scalability.** Partitioned regions can scale to large amounts of data because the data divided between the members available to host the region. Increase your data capacity dynamically by simply adding new members. Partitioned regions also allow you to scale your processing capacity. Because your entries are spread out across the members hosting the region, reads and writes to those entries are also spread out across those members.
- **Good write performance.** You can configure the number of copies of your data. The amount of data transmitted per write does not increase with the number of members. By contrast, with replicated regions, each write must be sent to every member that has the region replicated, so the amount of data transmitted per write increases with the number of members.

In partitioned regions, you can collocate keys within buckets and across multiple partitioned regions. You can also control which members store which data buckets.

## Replicated Regions



**Note:** Replication is a good choice for small to medium size server regions.

Replicated regions provide the highest performance in terms of throughput and latency.

Use replicated regions for:

- **Small amounts of data required by all members of the distributed system.** For example, currency rate information and mortgage rates.
- **Data sets that can be contained entirely in a single member.** Each replicated region holds the complete data set for the region

- **High performance data access.** Replication guarantees local access from the heap for application threads, providing the lowest possible latency for data access.
- **Asynchronous distribution.** All distributed regions, replicated and non-replicated, provide the fastest distribution speeds.

### Distributed, Non-Replicated Regions

Distributed regions provide the same performance as replicated regions, but each member only stores data it has expressed interest in, either by subscribing to events from other members or by defining the data entries in its cache.

Use distributed, non-replicated regions for:

- **Peer regions, but not server regions or client regions.** Server regions must be either replicated or partitioned. Client regions must be local.
- **Data sets where individual members only need notification and updates for changes to a subset of the data.** In non-replicated regions, each member only receives update events for the data entries it has defined in the local cache.
- **Asynchronous distribution.** All distributed regions, replicated and non-replicated, provide the fastest distribution speeds.

### Local Regions



**Note:** When created using the `ClientRegionShortcut` settings, client regions are automatically defined as local, since all client distribution activities go to and come from the server tier.

The local region has no peer-to-peer distribution activity.

Use local regions for:

- **Client regions.** Distribution is only between the client and server tier.
- **Private data sets for the defining member.** The local region is not visible to peer members.

## Region Data Stores and Data Accessors

Understand the difference between members that store data for a region and members that only act as data accessors to the region.

In most cases, when you define a data region in a member's cache, you also specify whether the member is a data store. Members that store data for a region are referred to as data stores or data hosts. Members that do not store data are referred to as accessor members, or empty members. Any member, store or accessor, that defines a region can access it, put data into it, and receive events from other members. To configure a region so the member is a data accessor, you use configurations that specify no local data storage for the region. Otherwise, the member is a data store for the region.

## Creating Regions Dynamically

You can dynamically create regions in your application code and automatically instantiate them on members of a distributed system.

If your application does not require partitioned regions, you can use the `com.gemstone.gemfire.cache.DynamicRegionFactory` class to dynamically create regions, or you can create them using the `<dynamic-region-factory>` element in the `cache.xml` file that defines the region. See [<dynamic-region-factory>](#) on page 867.

Pivotal Inc. recommends that you use functions to create regions dynamically in your application, as described in this topic.

For a complete discussion of using GemFire functions, see [Function Execution](#) on page 417. Functions use the `com.gemstone.gemfire.cache.execute.FunctionService` class.

For example, the following Java classes define and use a function for dynamic region creation:

The `CreateRegionFunction` class defines a function invoked on a server by a client using the `onServer()` method of the `FunctionService` class. This function call initiates region creation by putting an entry into the region attributes metadata region. The entry key is the region name and the value is the set of region attributes used to create the region.

```
#CreateRegionFunction.java

import com.gemstone.gemfire.cache.Cache;
import com.gemstone.gemfire.cache.CacheFactory;
import com.gemstone.gemfire.cache.DataPolicy;
import com.gemstone.gemfire.cache.Declarable;
import com.gemstone.gemfire.cache.Region;
import com.gemstone.gemfire.cache.RegionAttributes;
import com.gemstone.gemfire.cache.RegionFactory;
import com.gemstone.gemfire.cache.Scope;

import com.gemstone.gemfire.cache.execute.Function;
import com.gemstone.gemfire.cache.execute.FunctionContext;

import java.util.Properties;

public class CreateRegionFunction implements Function, Declarable {

    private final Cache cache;

    private final Region<String,RegionAttributes>
regionAttributesMetadataRegion;

    private static final String REGION_ATTRIBUTES_METADATA_REGION =
"__regionAttributesMetadata";

    public enum Status {SUCCESSFUL, UNSUCCESSFUL, ALREADY_EXISTS};

    public CreateRegionFunction() {
        this.cache = CacheFactory.getAnyInstance();
        this.regionAttributesMetadataRegion =
createRegionAttributesMetadataRegion();
    }

    public void execute(FunctionContext context) {
        Object[] arguments = (Object[]) context.getArguments();
        String regionName = (String) arguments[0];
        RegionAttributes attributes = (RegionAttributes) arguments[1];

        // Create or retrieve region
        Status status = createOrRetrieveRegion(regionName, attributes);

        // Return status
        context.getResultSender().lastResult(status);
    }

    private Status createOrRetrieveRegion(String regionName, RegionAttributes
attributes) {
        Status status = Status.SUCCESSFUL;
        Region region = this.cache.getRegion(regionName);
```

```

        if (region == null) {
            // Put the attributes into the metadata region. The afterCreate call
            // will actually create the region.
            this.regionAttributesMetadataRegion.put(regionName, attributes);

            // Retrieve the region after creating it
            region = this.cache.getRegion(regionName);
            if (region == null) {
                status = Status.UNSUCCESSFUL;
            }
        } else {
            status = Status.ALREADY_EXISTS;
        }
        return status;
    }

    private Region<String,RegionAttributes>
createRegionAttributesMetadataRegion() {
    Region<String, RegionAttributes> metaRegion =
this.cache.getRegion(REGION_ATTRIBUTES_METADATA_REGION);
    if (metaRegion == null) {
        RegionFactory<String, RegionAttributes> factory =
this.cache.createRegionFactory();
        factory.setDataPolicy(DataPolicy.REPLICATE);
        factory.setScope(Scope.DISTRIBUTED_ACK);
        factory.addCacheListener(new CreateRegionCacheListener());
        metaRegion = factory.create(REGION_ATTRIBUTES_METADATA_REGION);
    }
    return metaRegion;
}

public String getId() {
    return getClass().getSimpleName();
}

public boolean optimizeForWrite() {
    return false;
}

public boolean isHA() {
    return true;
}

public boolean hasResult() {
    return true;
}

public void init(Properties properties) {
}
}

```

The CreateRegionCacheListener class is a cache listener that implements two methods, afterCreate() and afterRegionCreate(). The afterCreate() method creates the region. The afterRegionCreate() method causes each new server to create all the regions defined in the metadata region.

#### #CreateRegionCacheListener.java

```
import com.gemstone.gemfire.cache.Cache;
```

```

import com.gemstone.gemfire.cache.CacheFactory;
import com.gemstone.gemfire.cache.Declarable;
import com.gemstone.gemfire.cache.EntryEvent;
import com.gemstone.gemfire.cache.Region;
import com.gemstone.gemfire.cache.RegionAttributes;
import com.gemstone.gemfire.cache.RegionEvent;
import com.gemstone.gemfire.cache.RegionExistsException;

import com.gemstone.gemfire.cache.util.CacheListenerAdapter;

import java.util.Map;
import java.util.Properties;

public class CreateRegionCacheListener extends
CacheListenerAdapter<String,RegionAttributes> implements Declarable {

    private Cache cache;

    public CreateRegionCacheListener() {
        this.cache = CacheFactory.getAnyInstance();
    }

    public void afterCreate(EntryEvent<String,RegionAttributes> event) {
        createRegion(event.getKey(), event.getNewValue());
    }

    public void afterRegionCreate(RegionEvent<String,RegionAttributes> event)
{
        Region<String,RegionAttributes> region = event.getRegion();
        for (Map.Entry<String,RegionAttributes> entry : region.entrySet()) {
            createRegion(entry.getKey(), entry.getValue());
        }
    }

    private void createRegion(String regionName, RegionAttributes attributes)
{
        if (this.cache.getLogger().fineEnabled()) {
            this.cache.getLogger().fine("CreateRegionCacheListener creating region
named: " + regionName + " with attributes: " + attributes);
        }
        try {
            Region region = this.cache.createRegion(regionName, attributes);
            if (this.cache.getLogger().fineEnabled()) {
                this.cache.getLogger().fine("CreateRegionCacheListener created: "
+ region);
            }
            System.out.println("CreateRegionCacheListener created: " + region);
        } catch (RegionExistsException e) {/* ignore */}
    }

    public void init(Properties p) {
    }
}

```



# Chapter 28

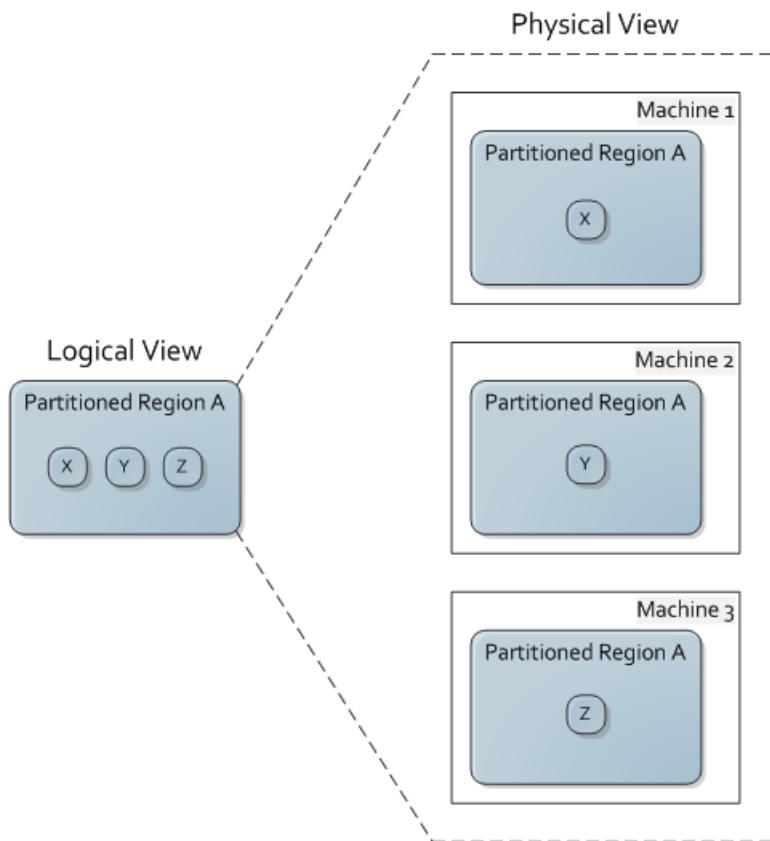
## Partitioned Regions

In addition to basic region management, partitioned regions include options for high availability, data location control, and data balancing across the distributed system.

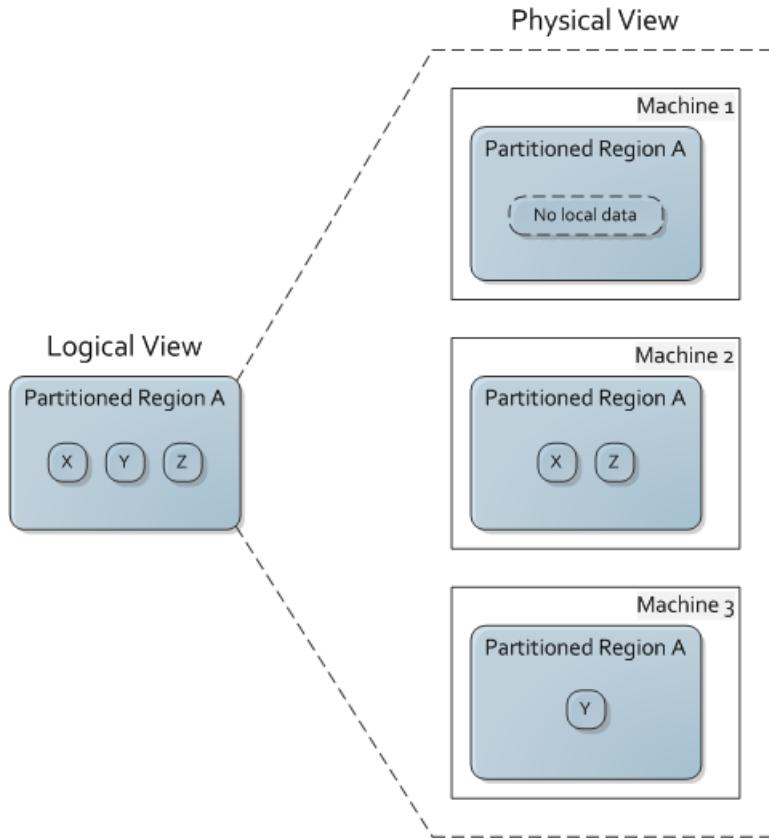
### Understanding Partitioning

To use partitioned regions, you should understand how they work and your options for managing them.

During operation, a partitioned region looks like one large virtual region, with the same logical view held by all of the members where the region is defined.



For each member where you define the region, you can choose how much space to allow for region data storage, including no local storage at all. The member can access all region data regardless of how much is stored locally.



A distributed system can have multiple partitioned regions, and it can mix partitioned regions with distributed regions and local regions. The usual requirement for unique region names, except for regions with local scope, still applies. A single member can host multiple partitioned regions.

### Data Partitioning

GemFire automatically determines the physical location of data in the members that host a partitioned region's data. GemFire breaks partitioned region data into units of storage known as buckets and stores each bucket in a region host member. Buckets are distributed in accordance to the member's region attribute settings.

When an entry is created, it is assigned to a bucket. Keys are grouped together in a bucket and always remain there. If the configuration allows, the buckets may be moved between members to balance the load.

You must run the data stores needed to accommodate storage for the partitioned region's buckets. You can start new data stores on the fly. When a new data store creates the region, it takes responsibility for as many buckets as allowed by the partitioned region and member configuration.

You can customize how GemFire groups your partitioned region data with custom partitioning and data colocation.

### Partitioned Region Operation

A partitioned region operates much like a non-partitioned region with distributed scope. Most of the standard Region methods are available, although some methods that are normally local operations become distributed operations, because they work on the partitioned region as a whole instead of the local cache. For example, a put or create into a partitioned region may not actually be stored into the cache of the member that called the operation. The retrieval of any entry requires no more than one hop between members.

Partitioned regions support the client/server model, just like other regions. If you need to connect dozens of clients to a single partitioned region, using servers greatly improves performance.

## Additional Information About Partitioned Regions

Keep the following in mind about partitioned regions:

- Partitioned regions never run asynchronously. Operations in partitioned regions always wait for acknowledgement from the caches containing the original data entry and any redundant copies.
- A partitioned region needs a cache loader in every region data store (`local-max-memory > 0`).
- GemFire distributes the data buckets as evenly as possible across all members storing the partitioned region data, within the limits of any custom partitioning or data colocation that you use. The number of buckets allotted for the partitioned region determines the granularity of data storage and thus how evenly the data can be distributed. The number of buckets is a total for the entire region across the distributed system.
- In rebalancing data for the region, GemFire moves buckets, but does not move data around inside the buckets.
- You can query partitioned regions, but there are certain limitations. See [Querying Partitioned Regions](#) on page 366 for more information.

## Configuring Partitioned Regions

Plan the configuration and ongoing management of your partitioned region for host and accessor members and configure the regions for startup.

Before you begin, understand [Basic Configuration and Programming](#) on page 131.

1. Start your region configuration using one of the PARTITION region shortcut settings. See [Region Shortcuts and Custom Named Region Attributes](#) on page 150.
2. If you need high availability for your partitioned region, configure for that. See [Configure High Availability for a Partitioned Region](#) on page 227.
3. Estimate the amount of space needed for the region. If you use redundancy, this is the max for all primary and secondary copies stored in the member. For example, with redundancy of one, each region data entry requires twice the space than with no redundancy, because the entry is stored twice. See [Memory Requirements for Cached Data](#) on page 989.
4. Configure the total number of buckets for the region. This number must be the same for colocated regions. See [Configuring the Number of Buckets for a Partitioned Region](#) on page 216.
5. Configure your members' data storage and data loading for the region:
  - a. You can have members with no local data storage and members with varying amounts of storage. Determine the max memory available in your different member types for this region. These will be set in the `partition-attributes local-max-memory`. This is the only setting in `partition-attributes` that can vary between members. Use these max values and your estimates for region memory requirements to help you figure how many members to start out with for the region.
  - b. For members that store data for the region (`local-max-memory` greater than 0), define a data loader. See [Implement a Data Loader](#) on page 255.
  - c. If you have members with no local data storage (`local-max-memory` set to 0), review your system startup/shutdown procedures. Make sure there is always at least one member with local data storage running when any members with no storage are running.
6. If you want to custom partition the data in your region or collocate data between multiple regions, code and configure accordingly. See [Understanding Custom Partitioning and Data Colocation](#) on page 218.
7. Plan your partition rebalancing strategy and configure and program for that. See [Rebalancing Partitioned Region Data](#) on page 230.

## Configuring the Number of Buckets for a Partitioned Region

Decide how many buckets to assign to your partitioned region and set the configuration accordingly.

The total number of buckets for the partitioned region determines the granularity of data storage and thus how evenly the data can be distributed. vFabric GemFire distributes the buckets as evenly as possible across the data stores. The number of buckets is fixed after region creation.

The partition attribute `total-num-buckets` sets the number for the entire partitioned region across all participating members. Set it using one of the following:

- XML:

```
<region name="PR1">
  <region-attributes refid="PARTITION">
    <partition-attributes total-num-buckets="7"/>
  </region-attributes>
</region>
```

- Java:

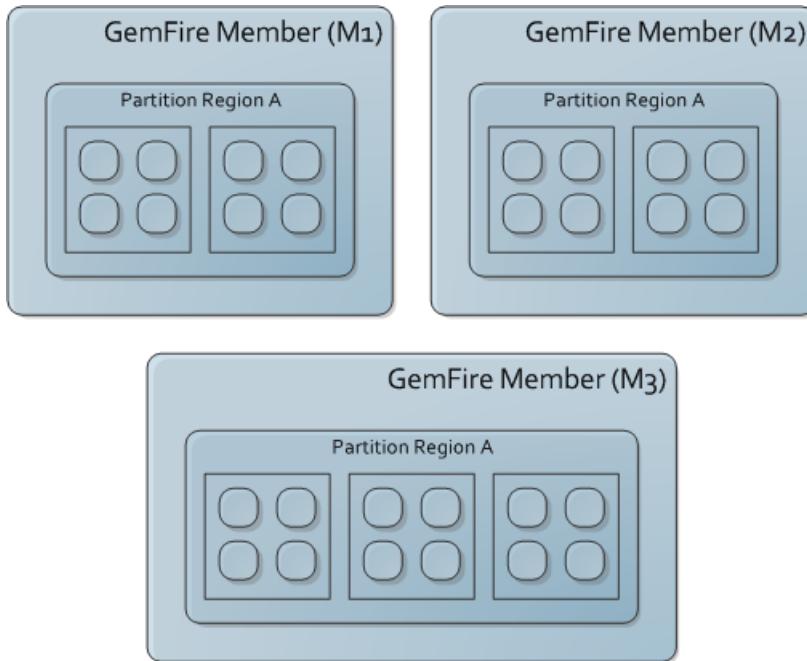
```
RegionFactory rf =
  cache.createRegionFactory(RegionShortcut.PARTITION);
rf.setPartitionAttributes(new
PartitionAttributesFactory().setTotalNumBuckets(7).create());
custRegion = rf.create("customer");
```

## Calculate the Total Number of Buckets for a Partitioned Region

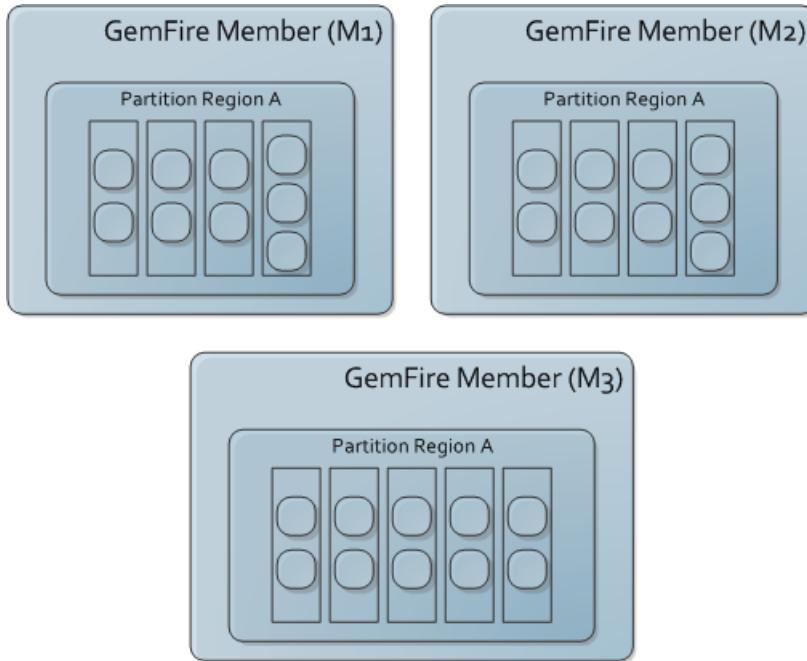
Follow these guidelines to calculate the total number of buckets for the partitioned region:

- Use a prime number. This provides the most even distribution.
- Make it at least four times as large as the number of data stores you expect to have for the region. The larger the ratio of buckets to data stores, the more evenly the load can be spread across the members. Note that there is a trade-off between load balancing and overhead, however. Managing a bucket introduces significant overhead, especially with higher levels of redundancy.

You are trying to avoid the situation where some members have significantly more data entries than others. For example, compare the next two figures. This figure shows a region with three data stores and seven buckets. If all the entries are accessed at about the same rate, this configuration creates a hot spot in member M3, which has about fifty percent more data than the other data stores. M3 is likely to be a slow receiver and potential point of failure.



Configuring more buckets gives you fewer entries in a bucket and a more balanced data distribution. This figure uses the same data as before but increases the number of buckets to 13. Now the data entries are distributed more evenly.



## Custom-Partitioning and Co-locating Data

You can customize how vFabric GemFire groups your partitioned region data with custom partitioning and data co-location.

## Understanding Custom Partitioning and Data Colocation

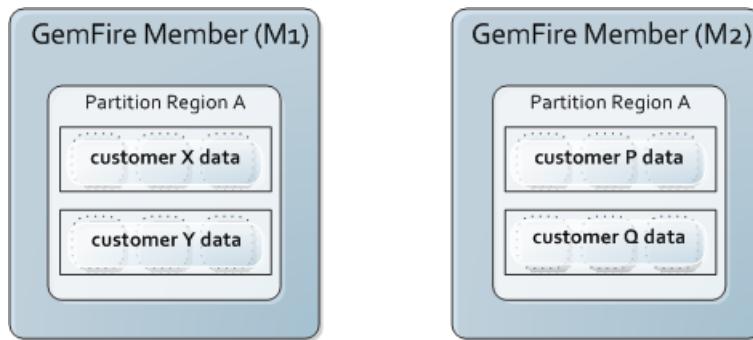
Custom partitioning and data colocation can be used separately or in conjunction with one another.

### Custom Partitioning

Use custom partitioning to group like entries into region buckets within a region. By default, GemFire assigns new entries to buckets based on the entry key contents. With custom partitioning, you can assign your entries to buckets in whatever way you want.

You can generally get better performance if you use custom partitioning to group similar data within a region. For example, a query run on all accounts created in January runs faster if all January account data is hosted by a single member. Grouping all data for a single customer can improve performance of data operations that work on customer data. Data aware function execution takes advantage of custom partitioning.

This figure shows a region with customer data that is grouped into buckets by customer.



With custom partitioning, you have two choices:

- **Standard custom partitioning.** With standard partitioning, you group entries into buckets, but you do not specify where the buckets reside. GemFire always keeps the entries in the buckets you have specified, but may move the buckets around for load balancing.
- **Fixed custom partitioning.** With fixed partitioning, you provide standard partitioning plus you specify the exact member where each data entry resides. You do this by assigning the data entry to a bucket and to a partition and by naming specific members as primary and secondary hosts of each partition.

This gives you complete control over the locations of your primary and any secondary buckets for the region. This can be useful when you want to store specific data on specific physical machines or when you need to keep data close to certain hardware elements.

Fixed partitioning has these requirements and caveats:

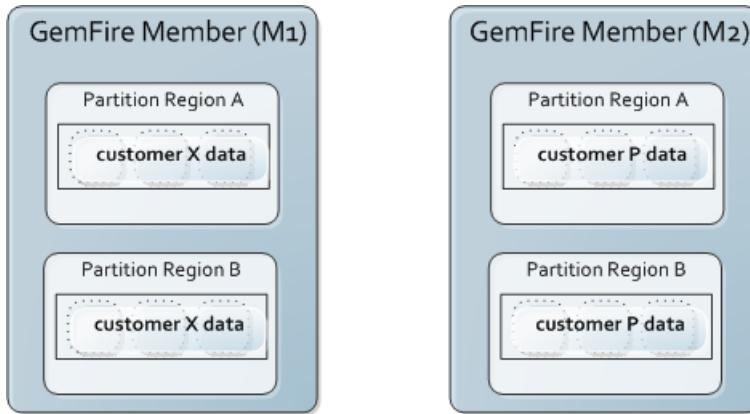
- GemFire cannot rebalance fixed partition region data because it cannot move the buckets around among the host members. You must carefully consider your expected data loads for the partitions you create.
- With fixed partitioning, the region configuration is different between host members. Each member identifies the named partitions it hosts, and whether it is hosting the primary copy or a secondary copy. You then program fixed partition resolver to return the partition id, so the entry is placed on the right members. Only one member can be primary for a particular partition name and that member cannot be the partition's secondary.

### Data Co-location Between Regions

With data co-location, GemFire stores entries that are related across multiple data regions in a single member. GemFire does this by storing all of the regions' buckets with the same ID together in the same member. During rebalancing operations, GemFire moves these bucket groups together or not at all.

So, for example, if you have one region with customer contact information and another region with customer orders, you can use co-location to keep all contact information and all orders for a single customer in a single member. This way, any operation done for a single customer uses the cache of only a single member.

This figure shows two regions with data co-location where the data is partitioned by customer type.



Data co-location requires the same data partitioning mechanism for all of the co-located regions. You can use the default partitioning provided by GemFire or custom partitioning.

You must use the same high availability settings across your co-located regions.

### Custom-Partition Your Region Data

By default, GemFire partitions each data entry into a bucket using a hashing policy on the key. Additionally, the physical location of the key-value pair is abstracted away from the application. You can change these policies for a partitioned region. You can provide your own data partitioning resolver and you can additionally specify which members host which data buckets.



**Note:** If you are collocating data between regions and custom partitioning the data in the regions, all collocated regions must use the same custom partitioning mechanism. See [Co-locate Data from Different Partitioned Regions](#) on page 223.

For standard partitioning, you use `com.gemstone.gemfire.cache.PartitionResolver`. To implement fixed partitioning, you use `com.gemstone.gemfire.cache.FixedPartitionResolver`.

#### Prerequisites

- Create partitioned regions. See [Understanding Partitioning](#) on page 213 and [Configuring Partitioned Regions](#) on page 215.
- Decide whether to use standard custom partitioning or fixed custom partitioning. See [Understanding Custom Partitioning and Data Colocation](#) on page 218.
- If you also want to colocate data from multiple regions, understand how to colocate. See [Co-locate Data from Different Partitioned Regions](#) on page 223.

#### Procedure

1. Using `com.gemstone.gemfire.cache.PartitionResolver` (standard partitioning) or `com.gemstone.gemfire.cache.FixedPartitionResolver` (fixed partitioning), implement the standard partitioning resolver or the fixed partitioning resolver in one of the following locations, listed here in the search order used by GemFire:
  - **Custom class.** You provide this class as the partition resolver to the region creation.
  - **Entry key.** You use the implementing key object for every operation on the region entries.

- **Cache callback argument.** This implementation restricts you to using methods that accept a cache callback argument to manage the region entries. For a full list of the methods that take a callback argument, see the Region Javadocs. Among the methods that you *cannot* use with cache callback are `putAll`, `getAll`, `containsKey`, and `getDistributedLock`. These methods do not work with the `FunctionService` execution calls.
2. If you need the resolver's `getName` method, program that.
  3. Program the resolver's `getRoutingObject` method to return the routing object for each entry, based on how you want to group the entries. Give the same routing object to entries you want to group together. GemFire will place the entries in the same bucket.

For example, here is an implementation on a region key object that groups the entries by month and year:

```
Public class TradeKey implements PartitionResolver
{
    private String tradeID;
    private Month month;
    private Year year;
    public TradingKey(){ }
    public TradingKey(Month month, Year year)
    {
        this.month = month;
        this.year = year;
    }
    public Serializable getRoutingObject(EntryOperation opDetails)
    {
        return this.month + this.year;
    }
}
```

4. For fixed partitioning only, program and configure additional fixed partitioning pieces:

- Set the fixed partition attributes for each member.

These attributes define the data stored for the region by the member and must be different for different members. See `com.gemstone.gemfire.cache.FixedPartitionAttributes` for definitions of the attributes. Define each partition-name in your data host members for the region. For each partition name, in the member you want to host the primary copy, define it with `is-primary` set to `true`. In every member you want to host the secondary copy, define it with `is-primary` set to `false` (the default). The number of secondaries must match the number of redundant copies you have defined for the region. See [Configure High Availability for a Partitioned Region](#) on page 227.



**Note:** Buckets for a partition are hosted only by the members that have defined the partition name in their `FixedPartitionAttributes`.

These examples set the partition attributes for a member to be the primary host for the "Q1" partition data and a secondary host for "Q3" partition data.

XML:

```
<cache>
    <region name="Trades">
        <region-attributes>
            <partition-attributes redundant-copies="1"
                partition-resolver= "QuarterFixedPartitionResolver">
                <fixed-partition-attributes partition-name="Q1"
                    is-primary="true"/>
                <fixed-partition-attributes partition-name="Q3"
                    is-primary="false" num-buckets="6"/>
            </partition-attributes>
        </region-attributes>
    </region>
</cache>
```

```

        </region>
</cache>

Java:

FixedPartitionAttribute fpa1 =
FixedPartitionAttributes.createFixedPartition("Q1", true);
FixedPartitionAttribute fpa3 =
FixedPartitionAttributes.createFixedPartition("Q3", false, 6);

PartitionAttributesFactory paf = new PartitionAttributesFactory()
    .setPartitionResolver(new QuarterFixedPartitionResolver())
    .setTotalNumBuckets(12)
    .setRedundantCopies(2)
    .addFixedPartitionAttribute(fpa1)
    .addFixedPartitionAttribute(fpa3);

AttributesFactory attrFactory = new AttributesFactory();
attrFactory.setPartitionAttributes(paf.create());

DistributedSystem system = DistributedSystem.connect(new Properties());
Cache cache = CacheFactory.create(system);
Region createRegion = cache.createRegion("Trades",
attrFactory.create());
Region<String, String> region = (PartitionedRegion)createRegion;

```

- b. Program the `FixedPartitionResolver` `getPartitionName` method to return the name of the partition for each entry, based on where you want the entries to reside. GemFire uses `getPartitionName` and `getRoutingObject` to determine where an entry is placed.



**Note:** To group entries, assign every entry in the group the same routing object and the same partition name.

This example places the data based on date, with a different partition name for each quarter-year and a different routing object for each month.

```

/**
 * Returns one of four different partition names
 * (Q1, Q2, Q3, Q4) depending on the entry's date
 */
class QuarterFixedPartitionResolver implements
    FixedPartitionResolver<String, String> {

    @Override
    public String getPartitionName(EntryOperation<String, String>
opDetails,
        Set<String> targetPartitions) {

        Date date = (Date)opDetails.getKey();
        Calendar cal = Calendar.getInstance();
        cal.setTime(date);
        int month = cal.get(Calendar.MONTH);
        if (month >= 0 && month < 3) {
            if (targetPartitions.contains("Q1")) return "Q1";
        }
        else if (month >= 3 && month < 6) {
            if (targetPartitions.contains("Q2")) return "Q2";
        }
        else if (month >= 6 && month < 9) {
            if (targetPartitions.contains("Q3")) return "Q3";
        }
    }
}

```

```

        else if (month >= 9 && month < 12) {
            if (targetPartitions.contains("Q4")) return "Q4";
        }
        return "Invalid Quarter";
    }

@Override
public String getName() {
    return "QuarterFixedPartitionResolver";
}

@Override
public Serializable getRoutingObject(EntryOperation<String, String>
opDetails) {
    Date date = (Date)opDetails.getKey();
    Calendar cal = Calendar.getInstance();
    cal.setTime(date);
    int month = cal.get(Calendar.MONTH);
    return month;
}

@Override
public void close() {
}
}

```

5. Configure or program so GemFire finds your resolver for every operation that you perform on the region's entries. How you do this depends on where you chose to program your custom partitioning implementation (step 1).
  - a. **Custom class.** Define the class for the region at creation. The resolver will be used for every entry operation. Use one of these methods:
    - XML:

```
<region name="trades">
    <region-attributes>
        <partition-attributes>
            <partition-resolver="TradesPartitionResolver">
                <class-name>myPackage.TradesPartitionResolver
                </class-name>
            </partition-resolver>
        <partition-attributes>
    </region-attributes>
</region>
```

    - Java:

```
PartitionResolver resolver = new TradesPartitionResolver();
PartitionAttributes attrs =
    new PartitionAttributesFactory()
    .setPartitionResolver(resolver).create();
Region Trades =
    new RegionFactory().setPartitionAttributes(attrs).create("trades");
```
  - b. **Entry key.** Use the key object with the resolver implementation for every entry operation.
  - c. **Cache callback argument.** Provide the argument to every call that accesses an entry. This restricts you to calls that take a callback argument.

- If your collocated data is in a server system, add the `PartitionResolver` implementation class to the CLASSPATH of your Java clients. The resolver is used for single hop access to partitioned region data in the servers.

### Co-locate Data from Different Partitioned Regions

By default, GemFire allocates the data locations for a partitioned region independent of the data locations for any other partitioned region. You can change this policy for any group of partitioned regions, so that cross-region, related data is all hosted by the same member. This co-location speeds queries and other operations that access data from the regions.



**Note:** If you are collocating data between regions and custom partitioning the data in the regions, all collocated regions must use the same custom partitioning mechanism. See [Custom-Partition Your Region Data](#) on page 219.

Data co-location between partitioned regions generally improves the performance of data-intensive operations. You can reduce network hops for iterative operations on related data sets. Compute-heavy applications that are data-intensive can significantly increase overall throughput. For example, a query run on a patient's health records, insurance, and billing information is more efficient if all data is grouped in a single member. Similarly, a financial risk analytical application runs faster if all trades, risk sensitivities, and reference data associated with a single instrument are together.

#### Prerequisites

- Understand how to configure and create your partitioned regions. See [Understanding Partitioning](#) on page 213 and [Configuring Partitioned Regions](#) on page 215.
- (Optional) Understand how to custom-partition your data. See [Custom-Partition Your Region Data](#) on page 219.
- (Optional) If you want your co-located regions to be highly available, understand how high availability for partitioned regions works. See [Understanding High Availability for Partitioned Regions](#) on page 225.
- (Optional) Understand how to persist your region data. See [Configure Region Persistence and Overflow](#) on page 247.

#### Procedure

- Identify one region as the central region, with which data in the other regions is explicitly co-located. If you use persistence for any of the regions, you must persist the central region.
  - Create the central region before you create the others, either in the cache.xml or your code. Regions in the XML are created before regions in the code, so if you create any of your co-located regions in the XML, you must create the central region in the XML before the others. GemFire will verify its existence when the others are created and return `IllegalStateException` if the central region is not there. Do not add any collocation specifications to this central region.
  - For all other regions, in the region partition attributes, provide the central region's name in the `colocated-with` attribute. Use one of these methods:

- XML:

```
<cache>
  <region name="trades">
    <region-attributes>
      <partition-attributes>
        ...
        <partition-attributes>
      </partition-attributes>
    </region-attributes>
  </region>
  <region name="trade_history">
    <region-attributes>
      <partition-attributes colocated-with="trades">
```

```

        ...
    <partition-attributes>
    </region-attributes>
</region>
</cache>
```

- Java:

```

PartitionAttributes attrs = ...
Region trades = new
RegionFactory().setPartitionAttributes(attrs).create("trades");
...
attrs = new
PartitionAttributesFactory().setColocatedWith(trades.getFullPath()).create();
Region trade_history = new
RegionFactory().setPartitionAttributes(attrs).create("trade_history");
```

2. For each of the co-located regions, use the same values for these partition attributes related to bucket management:

- recovery-delay
- redundant-copies
- startup-recovery-delay
- total-num-buckets

3. If you custom partition your region data, provide the same custom resolver to all co-located regions:

- XML:

```

<cache>
    <region name="trades">
        <region-attributes>
            <partition-attributes>
                <partition-resolver="TradesPartitionResolver">
                    <class-name>myPackage.TradesPartitionResolver
                    </class-name>
                <partition-attributes>
            </region-attributes>
        </region>
        <region name="trade_history">
            <region-attributes>
                <partition-attributes colocated-with="trades">
                    <partition-resolver="TradesPartitionResolver">
                        <class-name>myPackage.TradesPartitionResolver
                        </class-name>
                    <partition-attributes>
                </region-attributes>
            </region>
        </cache>
```

- Java:

```

PartitionResolver resolver = new TradesPartitionResolver();
PartitionAttributes attrs =
    new PartitionAttributesFactory()
        .setPartitionResolver(resolver).create();
Region trades = new
RegionFactory().setPartitionAttributes(attrs).create("trades");
attrs = new PartitionAttributesFactory()
```

```
.setColocatedWith(trades.getFullPath()).setPartitionResolver(resolver).create();
Region trade_history = new
RegionFactory().setPartitionAttributes(attrs).create("trade_history");
```

4. If you want to persist data in the co-located regions, persist the central region and then persist the other regions as needed. Use the same disk store for all of the co-located regions that you persist.

## Configuring High Availability for Partitioned Regions

By default, vFabric GemFire stores only a single copy of your partitioned region data among the region's data stores. You can configure GemFire to maintain redundant copies of your partitioned region data for high availability.

### Understanding High Availability for Partitioned Regions

With high availability, each member that hosts data for the partitioned region gets some primary copies and some redundant (secondary) copies.

With redundancy, if one member fails, operations continue on the partitioned region with no interruption of service:

- If the member hosting the primary copy is lost, GemFire makes a secondary copy the primary. This might cause a temporary loss of redundancy, but not a loss of data.
- Whenever there are not enough secondary copies to satisfy redundancy, the system works to recover redundancy by assigning another member as secondary and copying the data to it.



**Note:** You can still lose cached data when you are using redundancy if enough members go down in a short enough time span.

You can configure how the system works to recover redundancy when it is not satisfied. You can configure recovery to take place immediately or, if you want to give replacement members a chance to start up, you can configure a wait period. Redundancy recovery is also automatically attempted during any partitioned data rebalancing operation. Without redundancy, the loss of any of the region's data stores causes the loss of some of the region's cached data.

Generally, you should not use redundancy when your applications can directly read from another data source, or when write performance is more important than read performance.

### Controlling Where Your Primaries and Secondaries Reside

By default, GemFire places your primary and secondary data copies for you, avoiding placement of two copies on the same physical machine. If there are not enough machines to keep different copies separate, GemFire places copies on the same physical machine. You can change this behavior, so GemFire only places copies on separate machines.

You can also control which members store your primary and secondary data copies. GemFire provides two options:

- **Fixed custom partitioning.** This option is set for the region. Fixed partitioning gives you absolute control over where your region data is hosted. With fixed partitioning, you provide GemFire with the code that specifies the bucket and data store for each data entry in the region. When you use this option with redundancy, you specify the primary and secondary data stores. Fixed partitioning does not participate in rebalancing because all bucket locations are fixed by you.
- **Redundancy zones.** This option is set at the member level. Redundancy zones let you separate primary and secondary copies by member groups, or zones. You assign each data host to a zone. Then GemFire places redundant copies in different redundancy zones, the same as it places redundant copies on different physical machines. You can use this to split data copies across different machine racks or networks. This option allows

you to add members on the fly and use rebalancing to redistribute the data load, with redundant data maintained in separate zones. When you use redundancy zones, GemFire will not place two copies of the data in the same zone, so make sure you have enough zones.

## Running Processes in VMware Virtual Machines

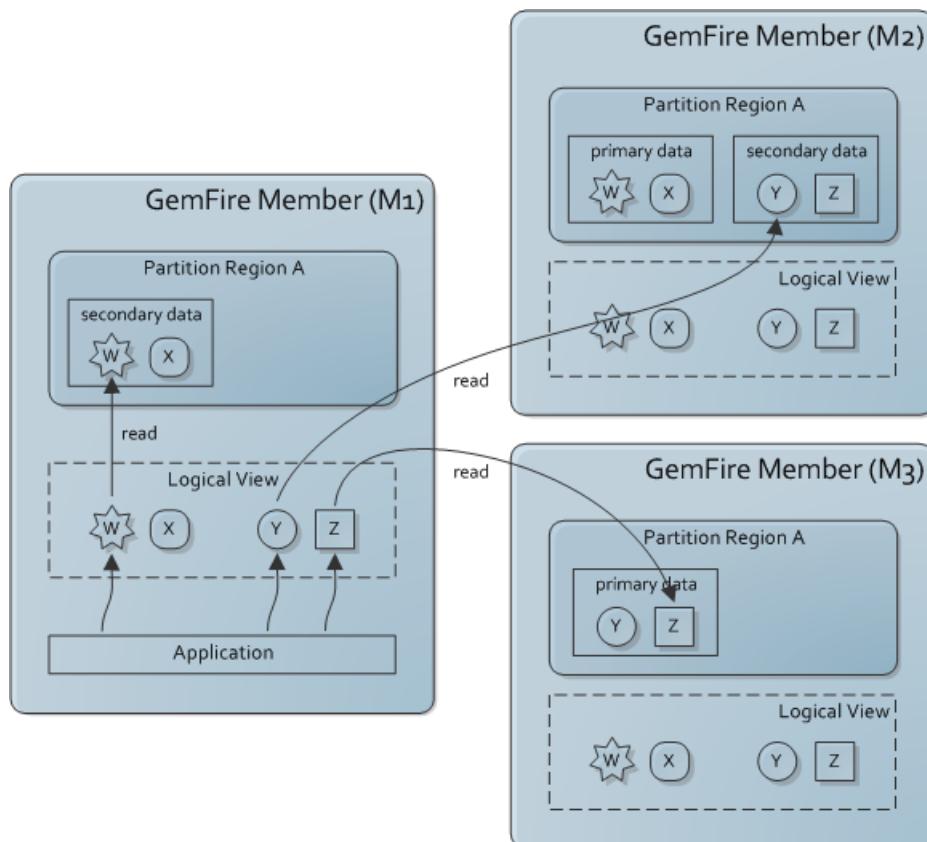
By default, GemFire stores redundant copies on different machines. When you run your processes in VMware virtual machines, the normal view of the machine becomes the VMware VM and not the physical machine. If you run multiple VMWare VMs on the same physical machine, you could end up storing partitioned region primary buckets in separate VMware VMs, but on the same physical machine as your secondaries. If the physical machine fails, you can lose data. When you run in VMware VMs, you can configure GemFire to identify the physical machine and store redundant copies on different physical machines.

## Reads and Writes in Highly-Available Partitioned Regions

GemFire treats reads and writes differently in highly-available partitioned regions than in other regions because the data is available in multiple members:

- Write operations (like `put` and `create`) go to the primary for the data keys and then are distributed synchronously to the redundant copies. Events are sent to the members configured with `subscription-attributes interest-policy` set to `all`.
- Read operations go to any member holding a copy of the data, with the local cache favored, so a read intensive system can scale much better and handle higher loads.

In this figure, M1 is reading W, Y, and Z. It gets W directly from its local copy. Since it doesn't have a local copy of Y or Z, it goes to a cache that does, picking the source cache at random.



## Configure High Availability for a Partitioned Region

Configure in-memory high availability for your partitioned region. Set other high-availability options, like redundancy zones and redundancy recovery strategies.

Here are the main steps for configuring high availability for a partitioned region. See later sections for details.

1. Set the number of redundant copies the system should maintain of the region data. See [Set the Number of Redundant Copies](#) on page 227.
2. (Optional) If you want to group your data store members into redundancy zones, configure them accordingly. See [Configure Redundancy Zones for Members](#) on page 227.
3. (Optional) If you want GemFire to only place redundant copies on different physical machines, configure for that. See [Set Enforce Unique Host](#) on page 228.
4. Decide how to manage redundancy recovery and change GemFire's default behavior as needed.
  - a) **After a member crashes.** If you want automatic redundancy recovery, change the configuration for that. See [Configure Member Crash Redundancy Recovery for a Partitioned Region](#) on page 228.
  - b) **After a member joins.** If you do *not* want immediate, automatic redundancy recovery, change the configuration for that. See [Configure Member Join Redundancy Recovery for a Partitioned Region](#) on page 229.
5. For all but fixed partitioned regions, review the points at which you kick off rebalancing. Redundancy recovery is done automatically at the start of any rebalancing. This is most important if you run with no automated recovery after member crashes or joins. See [Rebalancing Partitioned Region Data](#) on page 230.

During runtime, you can add capacity by adding new members for the region. For regions that do not use fixed partitioning, you can also kick off a rebalancing operation to spread the region buckets among all members.

### Set the Number of Redundant Copies

Configure in-memory high availability for your partitioned region by specifying the number of secondary copies you want to maintain in the region's data stores.

Specify the number of redundant copies you want for your partitioned region data in the partition attribute `redundant-copies` setting. The default setting is 0.

For example:

- XML:

```
<region name="PR1">
  <region-attributes refid="PARTITION">
    <partition-attributes redundant-copies="1"/>
  </region-attributes>
</region>
```

- Java:

```
PartitionAttributes pa =
  new
  PartitionAttributesFactory().setRedundantCopies(1).create();
```

### Configure Redundancy Zones for Members

Group members into redundancy zones so GemFire will separate redundant data copies into different zones.

Understand how to set a member's `gemfire.properties` settings. See [gemfire.properties](#) and [gfsecurity.properties \(GemFire Properties\)](#) on page 851.

Group your partition region hosts into redundancy zones with the `gemfire.properties` setting `redundancy-zone`.

For example, if you had redundancy set to 1, so you have one primary and one secondary copy of each data entry, you could split primary and secondary data copies between two machine racks by defining one redundancy zone for each rack. To do this, you set this zone in the `gemfire.properties` for all members that run on one rack:

```
redundancy-zone=rack1
```

You would set this zone `gemfire.properties` for all members on the other rack:

```
redundancy-zone=rack2
```

Each secondary copy would be hosted on the rack opposite the rack where its primary copy is hosted.

### **Set Enforce Unique Host**

Configure vFabric GemFire to use only unique physical machines for redundant copies of partitioned region data.

Understand how to set a member's `gemfire.properties` settings. See [gemfire.properties and gfsecurity.properties \(GemFire Properties\)](#) on page 851.

Configure your members so GemFire always uses different physical machines for redundant copies of partitioned region data using the `gemfire.properties` setting `enforce-unique-host`. The default for this setting is false.

Example:

```
enforce-unique-host=true
```

### **Configure Member Crash Redundancy Recovery for a Partitioned Region**

Configure whether and how redundancy is recovered in a partition region after a member crashes.

Use the partition attribute `recovery-delay` to specify member crash redundancy recovery.

recovery-delay partition attribute	Effect following a member failure
-1	No automatic recovery of redundancy following a member failure. This is the default.
long greater than or equal to 0	Number of milliseconds to wait after a member failure before recovering redundancy.

By default, redundancy is not recovered after a member crashes. If you expect to quickly restart most crashed members, combining this default setting with member join redundancy recovery can help you avoid unnecessary data shuffling while members are down. By waiting for lost members to rejoin, redundancy recovery is done using the newly started members and partitioning is better balanced with less processing.

Set crash redundancy recovery using one of the following:

- XML:

```
// Give a crashed member 10 seconds to restart
// before recovering redundancy
<region name="PR1">
  <region-attributes refid="PARTITION">
    <partition-attributes recovery-delay="10000" />
  </region-attributes>
</region>
```

- Java:

```
PartitionAttributes pa = new
PartitionAttributesFactory().setRecoveryDelay(10000).create();
```

### Configure Member Join Redundancy Recovery for a Partitioned Region

Configure whether and how redundancy is recovered in a partition region after a member joins.

Use the partition attribute `startup-recovery-delay` to specify member join redundancy recovery.

startup-recovery-delay partition attribute	Effect following a member join
-1	No automatic recovery of redundancy after a new member comes online. If you use this and the default <code>recovery-delay</code> setting, you can only recover redundancy by kicking off rebalancing through a cacheserver or API call.
long greater than or equal to 0	Number of milliseconds to wait after a member joins before recovering redundancy. The default is 0 (zero), which causes immediate redundancy recovery whenever a new partitioned region host joins.

Setting this to a value higher than the default of 0 allows multiple new members to join before redundancy recovery kicks in. With the multiple members present during recovery, the system will spread redundancy recovery among them. With no delay, if multiple members are started in close succession, the system may choose only the first member started for most or all of the redundancy recovery.



**Note:** Satisfying redundancy is not the same as adding capacity. If redundancy is satisfied, new members do not take buckets until you invoke a rebalance.

Set join redundancy recovery using one of the following:

- XML:

```
// Wait 5 seconds after a new member joins before
// recovering redundancy
<region name="PR1">
    <region-attributes refid="PARTITION">
        <partition-attributes startup-recovery-delay="5000"/>
    </region-attributes>
</region>
```

- Java:

```
PartitionAttributes pa = new
PartitionAttributesFactory().setStartupRecoveryDelay(5000).create();
```

## Configuring Single-Hop Client Access to Server-Partitioned Regions

Single-hop data access enables the client pool to track where a partitioned region's data is hosted in the servers. To access a single entry, the client directly contacts the server that hosts the key, in a single hop.

### Understanding Client Single-Hop Access to Server-Partitioned Regions

With single-hop access the client connects to every server, so more connections are generally used. This works fine for smaller installations, but is a barrier to scaling.

If you have a large installation with many clients, you may want to disable single hop by setting the pool attribute, `pr-single-hop-enabled` to false in your pool declarations.

Without single hop, the client uses whatever server connection is available, the same as with all other operations. The server that receives the request determines the data location and contacts the host, which might be a different server. So more multiple-hop requests are made to the server system.



**Note:** Single hop is only used for these operations: put, get, destroy, putAll.

Even with single hop access enabled, you will occasionally see some multiple-hop behavior. To perform single-hop data access, clients automatically get metadata from the servers about where the entry buckets are hosted. The metadata is maintained lazily. It is only updated after a single-hop operation ends up needing multiple hops, an indicator of stale metadata in the client.

### Single Hop and the Pool max-connections Setting

Do not set the pool's max-connections setting with single hop enabled. Limiting the pool's connections with single hop can cause connection thrashing, throughput loss, and server log bloat.

If you need to limit the pool's connections, either disable single hop or keep a close watch on your system for these negative effects.

Setting no limit on connections, however, can result in too many connections to your servers, possibly causing you to run up against your system's file handle limits. Review your anticipated connection use and make sure your servers are able to accommodate it.

### Balancing Single-Hop Server Connection Use

Single-hop gives the biggest benefits when data access is well balanced across your servers. In particular, the loads for client/server connections can get out of balance if you have these in combination:

- Servers that are empty data accessors or that do not host the data the clients access through single-key operations
- Many single-key operations from the clients

If data access is greatly out of balance, clients can thrash trying to get to the data servers. In this case, it might be faster to disable single hop and go through servers that do not host the data.

### Configure Client Single-Hop Access to Server-Partitioned Regions

Configure your client/server system for direct, single-hop access to partitioned region data in the servers.

This requires a client/server installation that uses one or more partitioned regions on the server.

1. Verify the client's pool attribute, pr-single-hop-enabled is not set or is set to true. It is true by default.
2. If possible, leave the pool's max-connections at the default unlimited setting (-1).
3. If possible, use a custom data resolver to partition your server region data according to your clients' data use patterns. See [Custom-Partition Your Region Data](#) on page 219. Include the server's partition resolver implementation in the client's CLASSPATH. The server passes the name of the resolver for each custom partitioned region, so the client uses the proper one. If the server does not use a partition resolver, the default partitioning between server and client matches, so single hop works.
4. Add single-hop considerations to your overall server load balancing plan. Single-hop uses data location rather than least loaded server to pick the servers for single-key operations. Poorly balanced single-hop data access can affect overall client/server load balancing. Some counterbalancing is done automatically because the servers with more single-key operations become more loaded and are less likely to be picked for other operations.

### Rebalancing Partitioned Region Data

In a distributed system with minimal contention to the concurrent threads reading or updating from the members, you can use rebalancing to dynamically increase or decrease your data and processing capacity.

Rebalancing is a member operation. It affects all partitioned regions defined by the member, regardless of whether the member hosts data for the regions. The rebalancing operation performs two tasks:

1. If the configured partition region redundancy is not satisfied, rebalancing does what it can to recover redundancy. See [Configure High Availability for a Partitioned Region](#) on page 227.
2. Rebalancing moves the partitioned region data buckets between host members as needed to establish the most fair balance of data and behavior across the distributed system.

For efficiency, when starting multiple members, trigger the rebalance a single time, after you have added all members.



**Note:** If you have transactions running in your system, be careful in planning your rebalancing operations. Rebalancing may move data between members, which could cause a running transaction to fail with a `TransactionDataRebalancedException`. Fixed custom partitioning prevents rebalancing altogether. All other data partitioning strategies allow rebalancing and can result in this exception unless you run your transactions and your rebalancing operations at different times.

Kick off a rebalance using one of the following:

- `gfsh` command. First, starting a `gfsh` prompt and connect to the GemFire distributed system. Then type the following command:

```
gfsh>rebalance
```

Optionally, you can specify regions to include or exclude from rebalancing, specify a time-out for the rebalance operation or just [How to Simulate Region Rebalancing](#) on page 232. Type `help rebalance` or see [rebalance](#) on page 705 for more information.

- API call:

```
ResourceManager manager = cache.getResourceManager();
RebalanceOperation op = manager.createRebalanceFactory().start();
//Wait until the rebalance is complete and then get the results
RebalanceResults results = op.getResults();
//These are some of the details we can get about the run from the API
System.out.println("Took " + results.getTotalTime() + " milliseconds\n");

System.out.println("Transferred " + results.getTotalBucketTransferBytes() +
" bytes\n");
```

You can also just simulate a rebalance through the API, to see if it's worth it to run:

```
ResourceManager manager = cache.getResourceManager();
RebalanceOperation op = manager.createRebalanceFactory().simulate();
RebalanceResults results = op.getResults();
System.out.println("Rebalance would transfer " +
results.getTotalBucketTransferBytes() + " bytes ");
System.out.println(" and create " + results.getTotalBucketCreatesCompleted() +
" buckets.\n");
```

## How Partitioned Region Rebalancing Works

The rebalancing operation runs asynchronously.

As a general rule, rebalancing is performed on one partitioned region at a time. For regions that have co-located data, the rebalancing works on the regions as a group, maintaining the data co-location between the regions.

You can continue to use your partitioned regions normally while rebalancing is in progress. Read operations, write operations, and function executions continue while data is moving. If a function is executing on a local data set, you may see a performance degradation if that data moves to another host during function execution. Future function invocations are routed to the correct member.

GemFire tries to ensure that each member has the same percentage of its available space used for each partitioned region. The percentage is configured in the `partition-attributes local-max-memory` setting.

Partitioned region rebalancing:

- Does not allow the `local-max-memory` setting to be exceeded unless LRU eviction is enabled with overflow to disk.
- Places multiple copies of the same bucket on different host IP addresses whenever possible.
- Resets entry time to live and idle time statistics during bucket migration.
- Replaces offline members.

### When to Rebalance a Partitioned Region

You typically want to trigger rebalancing when capacity is increased or reduced through member startup, shutdown or failure.

You may also need to rebalance when:

- You use redundancy for high availability and have configured your region to not automatically recover redundancy after a loss. In this case, GemFire only restores redundancy when you invoke a rebalance. See [Configure High Availability for a Partitioned Region](#) on page 227.
- You have uneven hashing of data. Uneven hashing can occur if your keys do not have a hash code method, which ensures uniform distribution, or if you use a `PartitionResolver` to collocate your partitioned region data (see [Co-locate Data from Different Partitioned Regions](#) on page 223). In either case, some buckets may receive more data than others. Rebalancing can be used to even out the load between data stores by putting fewer buckets on members that are hosting large buckets.

### How to Simulate Region Rebalancing

You can simulate the rebalance operation before moving any actual data around by executing the `rebalance` command with the following option:

```
gfsh>rebalance --simulate
```



**Note:** If you are using `heap_lru` for data eviction, you may notice a difference between your simulated results and your actual rebalancing results. This discrepancy can be due to the VM starting to evict entries after you execute the simulation. Then when you perform an actual rebalance operation, the operation will make different decisions based on the newer heap size.

# Chapter 29

## Distributed and Replicated Regions

In addition to basic region management, distributed and replicated regions include options for things like push and pull distribution models, global locking, and region entry versions to ensure consistency across GemFire members.

### How Distribution Works

To use distributed and replicated regions, you should understand how they work and your options for managing them.



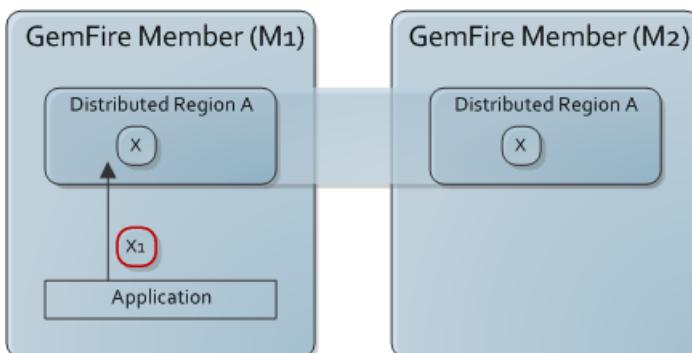
**Note:** The management of replicated and distributed regions supplements the general information for managing data regions provided in [Basic Configuration and Programming](#) on page 131. See also `com.gemstone.gemfire.cache.PartitionAttributes`.

A distributed region automatically sends entry value updates to remote caches and receives updates from them.

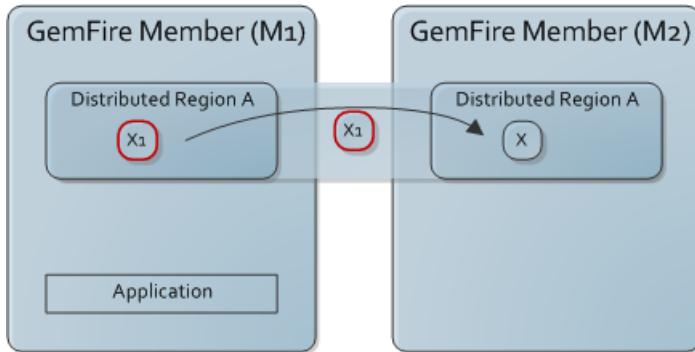
- Distributed entry updates come from the Region `put` and `create` operations (the creation of an entry with a non-null value is seen as an update by remote caches that already have the entry key). Entry updates are distributed selectively - only to caches where the entry key is already defined. This provides a pull model of distribution, compared to the push model that you get with replication.
- Distribution alone does not cause new entries to be copied from remote caches.
- A distributed region shares cache loader and cache writer application event handler plug-ins across the distributed system.

In a distributed region, new and updated entry values are automatically distributed to remote caches that already have the entries defined.

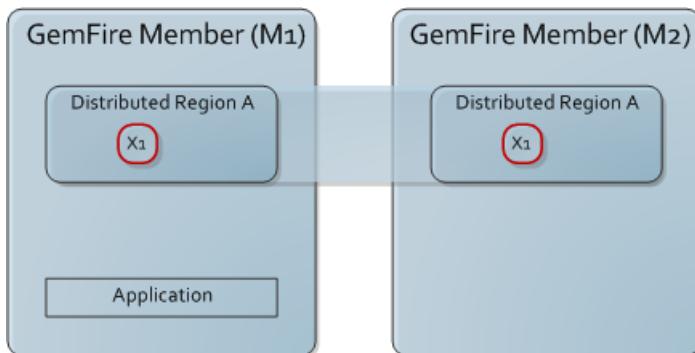
- **Step 1:** The application updates or creates the entry. At this point, the entry in the M1 cache may not yet exist.



- **Step 2:** The new value is automatically distributed to caches holding the entry.



- **Step 3:** The entry's value is the same throughout the distributed system.



## Options for Region Distribution

You can use distribution with and without acknowledgment, or global locking for your region distribution. Regions that are configured for distribution with acknowledgment can also be configured to resolve concurrent updates consistently across all GemFire members that host the region.

Each distributed region must have the same scope and concurrency checking setting throughout the distributed system.

Distributed scope is provided at three levels:

- **distributed-no-ack.** Distribution operations return without waiting for a response from other caches. This scope provides the best performance and uses the least amount of overhead, but it is also most prone to having inconsistencies caused by network problems. For example, a temporary disruption of the network transport layer could cause a failure in distributing updates to a cache on a remote machine, while the local cache continues being updated.
- **distributed-ack.** Distribution waits for acknowledgment from other caches before continuing. This behavior can be modified with the `early-ack` region attribute. This is slower than `distributed-no-ack`, but covers simple communication problems such as temporary network disruptions.

In systems where there are many `distributed-no-ack` operations, it is possible for `distributed-ack` operations to take a long time to complete. The distributed system has a configurable time to wait for acknowledgment to any `distributed-ack` message before sending alerts to the logs about a possible problem with the unresponsive member. No matter how long the wait, the sender keeps waiting in order to honor the `distributed-ack` region setting. The `gemfire.properties` attribute governing this is `ack-wait-threshold`.

- **global.** Entries and regions are automatically locked across the distributed system during distribution operations. All load, create, put, invalidate, and destroy operations on the region and its entries are performed with a distributed lock. The global scope enforces strict consistency across the distributed system, but it is the slowest mechanism for achieving consistency. In addition to the implicit locking performed by distribution operations, regions with global scope and their contents can be explicitly locked through the application APIs. This allows applications to perform atomic, multi-step operations on regions and region entries.

## How Replication and Preloading Work

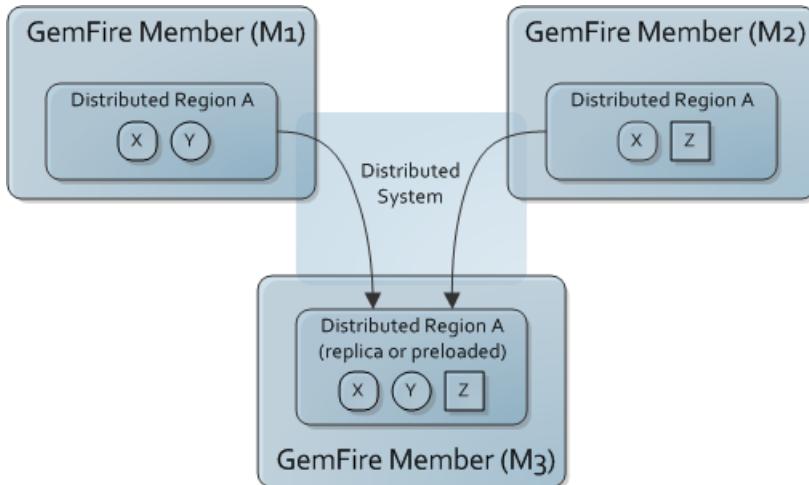
To work with replicated and preloaded regions, you should understand how their data is initialized and maintained in the cache.

Replicated and preloaded regions are configured by using one of the `REPLICATE` region shortcut settings, or by setting the region attribute `data-policy` to `replicate`, `persistent-replicate`, or `preloaded`.

### Initialization of Replicated and Preloaded Regions

At region creation, the system initializes the preloaded or replicated region with the most complete and up-to-date data set it can find. The system uses these data sources to initialize the new region, following this order of preference:

1. Another replicated region that is already defined in the distributed system.
2. For persistent replicate only. Disk files, followed by a union of all copies of the region in the distributed cache.
3. For preloaded region only. Another preloaded region that is already defined in the distributed system.
4. The union of all copies of the region in the distributed cache.

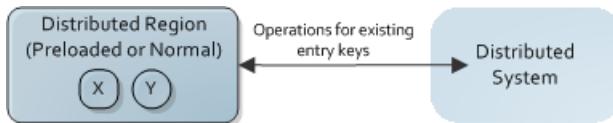


While a region is being initialized from a replicated or preloaded region, if the source region crashes, the initialization starts over.

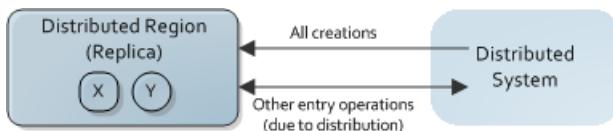
If a union of regions is used for initialization, as in the figure, and one of the individual source regions goes away during the initialization (due to cache closure, member crash, or region destruction), the new region may contain a partial data set from the crashed source region. When this happens, there is no warning logged or exception thrown. The new region still has a complete set of the remaining members' regions.

## Behavior of Replicated and Preloaded Regions After Initialization

Once initialized, the preloaded region operates like the region with a normal data-policy, receiving distributions only for entries it has defined in the local cache.



If the region is configured as a replicated region, it receives all new creations in the distributed region from the other members. This is the push distribution model. Unlike the preloaded region, the replicated region has a contract that states it will hold all entries that are present anywhere in the distributed region.



## Configure Distributed, Replicated, and Preloaded Regions

Plan the configuration and ongoing management of your distributed, replicated, and preloaded regions, and configure the regions.

Before you begin, understand [Basic Configuration and Programming](#) on page 131.

1. Choose the region shortcut setting that most closely matches your region configuration. See `com.gemstone.gemfire.cache.RegionShortcut`. To create a replicated region, use one of the REPLICATE shortcut settings. To create a preloaded region, set your region data-policy to preloaded. This cache.xml declaration creates a replicated region:

```
<region-attributes refid="REPLICATE">
</region-attributes>
```

2. Choose the level of distribution for your region. The region shortcuts in RegionShortcut for distributed regions use distributed-ack scope. If you need a different scope, set the region-attributes scope to distributed-no-ack or global.

Example:

```
<region-attributes refid="REPLICATE" scope="distributed-no-ack">
</region-attributes>
```

3. If you are using the distributed-ack scope, optionally enable concurrency checks for the region.

Example:

```
<region-attributes refid="REPLICATE" scope="distributed-ack"
concurrency-checks-enable="true">
</region-attributes>
```

4. If you are using global scope, program any explicit locking you need in addition to the automated locking provided by GemFire.

## Local Destroy and Invalidate in the Replicated Region

Of all the operations that affect the local cache only, only local region destroy is allowed in a replicated region. Other operations are not configurable or throw exceptions. For example, you cannot use local destroy as the expiration action on a replicated region. This is because local operations like entry invalidation and destruction remove data from the local cache only. A replicated region would no longer be complete if data were removed locally but left intact.

## Locking in Global Regions

In global regions, the system locks entries and the region during updates. You can also explicitly lock the region and its entries as needed by your application. Locking includes system settings that help you optimize performance and locking behavior between your members.

In regions with global scope, locking helps ensure cache consistency.

Locking of regions and entries is done in two ways:

1. **Implicit.** GemFire automatically locks global regions and their data entries during most operations. Region invalidation and destruction do not acquire locks.
2. **Explicit.** You can use the API to explicitly lock the region and its entries. Do this to guarantee atomicity in tasks with multi-step distributed operations. The Region methods `com.gemstone.gemfire.cache.Region.getDistributedLock` and `com.gemstone.gemfire.cache.Region.getRegionDistributedLock` return instances of `java.util.concurrent.locks.Lock` for a region and a specified key.



**Note:** You must use the Region API to lock regions and region entries. Do not use the `DistributedLockService` in the `com.gemstone.gemfire.distributed` package. That service is available only for locking in arbitrary distributed applications. It is not compatible with the Region locking methods.

## Lock Timeouts

Getting a lock on a region or entry is a two-step process of getting a lock instance for the entity and then using the instance to set the lock. Once you have the lock, you hold it for your operations, then release it for someone else to use. You can set limits on the time spent waiting to get a lock and the time spent holding it. Both implicit and explicit locking operations are affected by the timeouts:

- The lock timeout limits the wait to get a lock. The cache attribute `lock-timeout` governs implicit lock requests. For explicit locking, specify the wait time through your calls to the instance of `java.util.concurrent.locks.Lock` returned from the Region API. You can wait a specific amount of time, return immediately either with or without the lock, or wait indefinitely.

```
<cache lock-timeout="60">
</cache>
```

- The lock lease limits how long a lock can be held before it is automatically released. A timed lock allows the application to recover when a member fails to release an obtained lock within the lease time. For all locking, this timeout is set with the cache attribute `lock-lease`.

```
<cache lock-lease="120">
</cache>
```

## Optimize Locking Performance

For each global region, one of the members with the region defined will be assigned the job of lock grantor. The lock grantor runs the lock service that receives lock requests from system members, queues them as needed, and grants them in the order received.

The lock grantor is at a slight advantage over other members as it is the only one that does not have to send a message to request a lock. The grantor's requests cost the least for the same reason. Thus, you can optimize locking in a region by assigning lock grantor status to the member that acquires the most locks. This may be the member that performs the most puts and thus requires the most implicit locks or this may be the member that performs many explicit locks.

The lock grantor is assigned as follows:

- Any member with the region defined that requests lock grantor status is assigned it. Thus at any time, the most recent member to make the request is the lock grantor.
- If no member requests lock grantor status for a region, or if the current lock grantor goes away, the system assigns a lock grantor from the members that have the region defined in their caches.

You can request lock grantor status:

1. At region creation through the `is-lock-grantor` attribute. You can retrieve this attribute through the region method, `getAttributes`, to see whether you requested to be lock grantor for the region.



**Note:** The `is-lock-grantor` attribute does not change after region creation.

2. After region creation through the region `becomeLockGrantor` method. Changing lock grantors should be done with care, however, as doing so takes cycles from other operations. In particular, be careful to avoid creating a situation where you have members vying for lock grantor status.

## Examples

These two examples show entry locking and unlocking. Note how the entry's Lock object is obtained and then its lock method invoked to actually set the lock. The example program stores the entry lock information in a hash table for future reference.

```
/* Lock a data entry */
HashMap lockedItemsMap = new HashMap();
...
String entryKey = ...
if (!lockedItemsMap.containsKey(entryKey))
{
    Lock lock = this.currRegion.getDistributedLock(entryKey);
    lock.lock();
    lockedItemsMap.put(name, lock);
}
...
/* Unlock a data entry */
String entryKey = ...
if (lockedItemsMap.containsKey(entryKey))
{
    Lock lock = (Lock) lockedItemsMap.remove(name);
    lock.unlock();
}
```

# Chapter 30

---

## Consistency for Region Updates

GemFire ensures that all copies of a region eventually reach a consistent state on all members and clients that host the region, including GemFire members that distribute region events across a WAN.

### Consistency Checking by Region Type

GemFire performs different consistency checks depending on the type of region you have configured.

- [Partitioned Region Consistency](#) on page 239
- [Replicated Region Consistency](#) on page 239
- [Consistency for WAN Deployments](#) on page 240
- [Non-Replicated Region and Client Cache Consistency](#) on page 240

### Partitioned Region Consistency

For a partitioned region, GemFire maintains consistency by routing all updates on a given key to the GemFire member that holds the primary copy of that key. That member holds a lock on the key while distributing updates to other members that host a copy of the key. Because all updates to a partitioned region are serialized on the primary GemFire member, all members apply the updates in the same order and consistency is maintained at all times. See [Understanding Partitioning](#) on page 213.

### Replicated Region Consistency

For a replicated region, any member that hosts the region can update a key and distribute that update to other members without locking the key. It is possible that two members can update the same key at the same time (a concurrent update). It is also possible that, due to network latency, an update in one member is distributed to other members at a later time, after those members have already applied more recent updates to the key (an out-of-order update). By default, GemFire members perform conflict checking before applying region updates in order to detect and consistently resolve concurrent and out-of-order updates. Conflict checking ensures that region data eventually becomes consistent on all members that host the region. The conflict checking behavior for replicated regions is summarized as follows:

- If two members update the same key at the same time, conflict checking ensures that all members eventually apply the same value, which is the value of one of the two concurrent updates.
- If a member receives an out-of-order update (an update that is received after one or more recent updates were applied), conflict checking ensures that the out-of-order update is discarded and not applied to the cache.

[How Consistency Checking Works for Replicated Regions](#) on page 241 and [How Destroy and Clear Operations Are Resolved](#) on page 243 provide more details about how GemFire performs conflict checking when applying an update.

## Consistency for WAN Deployments

Regions can also be configured to distribute updates to other GemFire clusters over a WAN. With a distributed WAN configuration, multiple gateway senders asynchronously queue and send region updates to another GemFire cluster. It is possible for multiple sites to send updates to the same region entry at the same time. It is also possible that, due to a slow WAN connection, a cluster might receive region updates after a considerable delay, and after it has applied more recent updates to a region. To ensure that WAN-replicated regions eventually reach a consistent state, GemFire first ensures that each cluster performs consistency checking to regions before queuing updates to a gateway sender for WAN distribution. In other words, region conflicts are first detected and resolved in the local cluster, using the techniques described in the previous sections.

When a GemFire cluster in a WAN configuration receives a distributed update, conflict checking is performed to ensure that all sites apply updates in the same way. This ensures that regions eventually reach a consistent state across all GemFire clusters. The default conflict checking behavior for WAN-replicated regions is summarized as follows:

- If an update is received from the same GemFire cluster that last updated the region entry, then there is no conflict and the update is applied.
- If an update is received from a different GemFire cluster than the one that last updated the region entry, then a potential conflict exists. A cluster applies the update only when the update has a timestamp that is later than the timestamp currently recorded in the cache.



**Note:** If you use the default conflict checking feature for WAN deployments, you must ensure that all GemFire members in all clusters synchronize their system clocks. For example, use a common NTP server for all GemFire members that participate in a WAN deployment.

As an alternative to the default conflict checking behavior for WAN deployments, you can develop and deploy a custom conflict resolver for handling region events that are distributed over a WAN. Using a custom resolver enables you to handle conflicts using criteria other than, or in addition to, timestamp information. For example, you might always prioritize updates that originate from a particular site, given that the timestamp value is within a certain range.

See [How Consistency Is Achieved in WAN Deployments](#) on page 244 and, to configure a custom resolver, see "Implementing a `GatewayConflictResolver`" under [Resolving Conflicting Events](#) on page 200.

## Non-Replicated Region and Client Cache Consistency

When a member receives an update for an entry in a non-replicated region and applies an update, it performs conflict checking in the same way as for a replicated region. However, if the member initiates an operation on an entry that is not present in the region, it first passes that operation to a member that hosts a replicate. The member that hosts the replica generates and provides the version information necessary for subsequent conflict checking. See [How Consistency Checking Works for Replicated Regions](#) on page 241.

Client caches also perform consistency checking in the same way when they receive an update for a region entry. However, all region operations that originate in the client cache are first passed onto an available GemFire server, which generates the version information necessary for subsequent conflict checking.

## Configuring Consistency Checking

GemFire enables consistency checking by default. You cannot disable consistency checking for persistent regions. For all other regions, you can explicitly enable or disable consistency checking by setting the `concurrency-checks-enabled` region attribute in `cache.xml` to "true" or "false."

All GemFire members that host a region must use the same `concurrency-checks-enabled` setting for that region.

A client cache can disable consistency checking for a region even if server caches enable consistency checking for the same region. This configuration ensures that the client sees all events for the region, but it does not prevent the client cache region from becoming out-of-sync with the server cache.

See [<region-attributes>](#) on page 882.



**Note:** Regions that do not enable consistency checking remain subject to race conditions. Concurrent updates may result in one or more members having different values for the same key. Network latency can result in older updates being applied to a key after more recent updates have occurred.

## Overhead for Consistency Checks

Consistency checking requires additional overhead for storing and distributing version and timestamp information, as well as for maintaining destroyed entries for a period of time to meet consistency requirements.

To provide consistency checking, each region entry uses an additional 16 bytes. When an entry is deleted, a tombstone entry of approximately 13 bytes is created and maintained until the tombstone expires or is garbage-collected in the member. (When an is destroyed, the member temporarily retains the entry with its current version stamp to detect possible conflicts with operations that have occurred. The retained entry is referred to as a tombstone.) See [How Destroy and Clear Operations Are Resolved](#) on page 243.

If you cannot support the additional overhead in your deployment, you can disable consistency checks by setting `concurrency-checks-enabled` to "false" for each region. See [Consistency for Region Updates](#) on page 239.

## How Consistency Checking Works for Replicated Regions

Each region stores version and timestamp information for use in conflict detection. GemFire members use the recorded information to detect and resolve conflicts consistently before applying a distributed update.

By default, each entry in a region stores the ID of the GemFire member that last updated the entry, as well as a version stamp for the entry that is incremented each time an update occurs. The version information is stored in each local entry, and the version stamp is distributed to other GemFire members when the local entry is updated.

A GemFire member or client that receives an update message first compares the update version stamp with the version stamp recorded in its local cache. If the update version stamp is larger, it represents a newer version of the entry, so the receiving member applies the update locally and updates the version information. A smaller update version stamp indicates an out-of-order update, which is discarded.

An identical version stamp indicates that multiple GemFire members updated the same entry at the same time. To resolve a concurrent update, a GemFire member always applies (or keeps) the region entry that has the highest membership ID; the region entry having the lower membership ID is discarded.

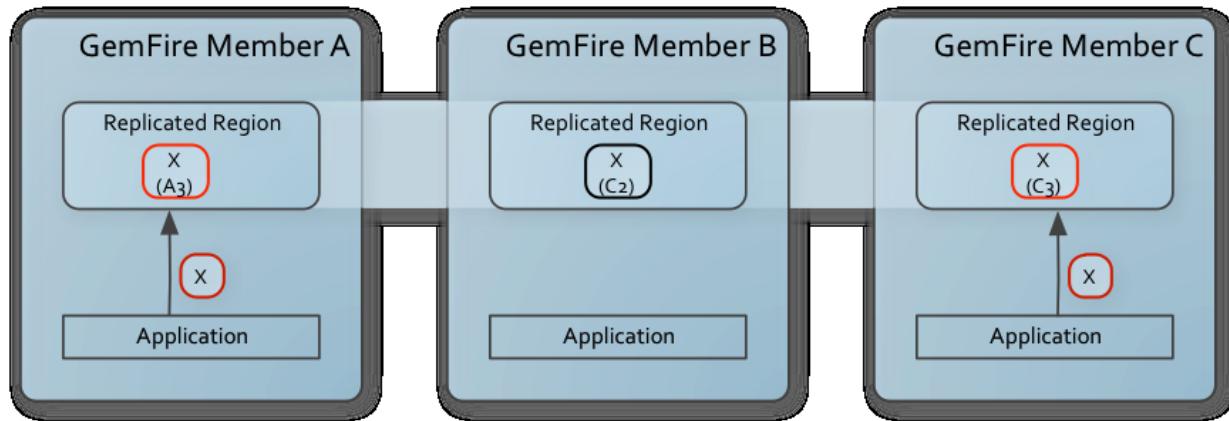


**Note:** When a GemFire member discards an update message (either for an out-of-order update or when resolving a concurrent update), it does not pass the discarded event to an event listener for the region, or to any configured gateway senders for WAN distribution. You can track the number of discarded updates for each member using the `conflatedEvents` statistic. See [GemFire Statistics List](#) on page 555. Some members may discard an update while other members apply the update, depending on the order in which each member receives the update. For this reason, the `conflatedEvents` statistic differs for each GemFire member. The example below describes this behavior in more detail.

The following example shows how a concurrent update is handled in a distributed system of three GemFire members. Assume that Members A, B, and C have membership IDs of 1, 2, and 3, respectively. Each member currently stores an entry, X, in their caches at version C2 (the entry was last updated by member C):

- **Step 1:** An application updates entry X on GemFire member A at the same time another application updates entry X on member C. Each member increments the version stamp for the entry and records the version stamp

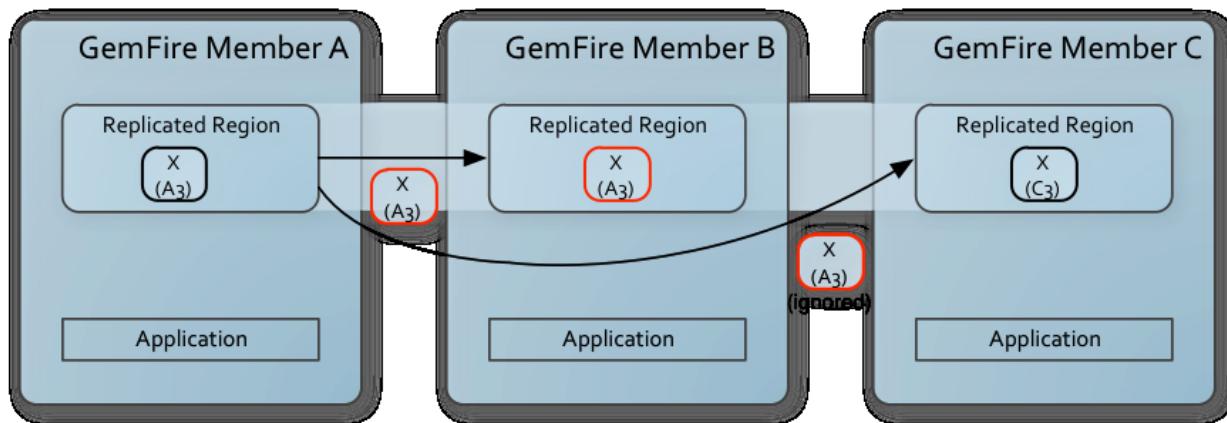
with their member ID in their local caches. In this case the entry was originally at version C2, so each member updates the version to 3 (A3 and C3, respectively) in their local caches.



- **Step 2:** Member A distributes its update message to members B and C.

Member B compares the update version stamp (3) to its recorded version stamp (2) and applies the update to its local cache as version A3. In this member, the update is applied for the time being, and passed on to configured event listeners.

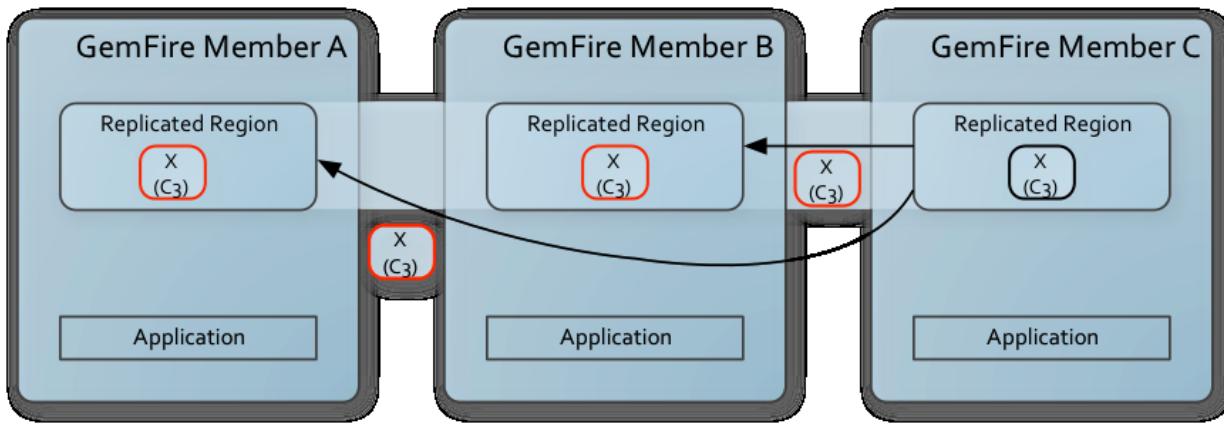
Member C compares the update version stamp (3) to its recorded version stamp (3) and identifies a concurrent update. To resolve the conflict, member C next compares the membership ID of the update to the membership ID stored in its local cache. Because the distributed system ID of the update (A3) is lower than the ID stored in the cache (C3), member C discards the update (and increments the conflatedEvents statistic).



- **Step 3:** Member C distributes the update message to members A and B.

Members A and B compare the update version stamp (3) to their recorded version stamps (3) and identify the concurrent update. To resolve the conflict, both members compare the membership ID of the update with the membership ID stored in their local caches. Because the distributed system ID of A in the cache value is less than the ID of C in the update, both members record the update C3 in their local caches, overwriting the previous value.

At this point, all members that host the region have achieved a consistent state for the concurrent updates on members A and C.



## How Destroy and Clear Operations Are Resolved

When consistency checking is enabled for a region, a GemFire member does not immediately remove an entry from the region when an application destroys the entry. Instead, the member retains the entry with its current version stamp for a period of time in order to detect possible conflicts with operations that have occurred. The retained entry is referred to as a *tombstone*. GemFire retains tombstones for partitioned regions and non-replicated regions as well as for replicated regions, in order to provide consistency.

A tombstone in a client cache or a non-replicated region expires after 8 minutes, at which point the tombstone is immediately removed from the cache.

A tombstone for a replicated or partitioned region expires after 10 minutes. Expired tombstones are eligible for garbage collection by the GemFire member. Garbage collection is automatically triggered after 100,000 tombstones of any type have timed out in the local GemFire member. You can optionally set the `gemfire.tombstone-gc-threshold` property to a value smaller than 100000 to perform garbage collection more frequently.



**Note:** To avoid out-of-memory errors, a GemFire member also initiates garbage collection for tombstones when the amount of free memory drops below 30 percent of total memory.

You can monitor the total number of tombstones in a cache using the `tombstoneCount` statistic in `CachePerfStats`. The `tombstoneGCCount` statistic records the total number of tombstone garbage collection cycles that a member has performed. `replicatedTombstonesSize` and `nonReplicatedTombstonesSize` show the approximate number of bytes that are currently consumed by tombstones in replicated or partitioned regions, and in non-replicated regions, respectively. See [GemFire Statistics List](#) on page 555.

## About Region.clear() Operations

Region entry version stamps and tombstones ensure consistency only when individual entries are destroyed. A `Region.clear()` operation, however, operates on all entries in a region at once. To provide consistency for `Region.clear()` operations, GemFire obtains a distributed read/write lock for the region, which blocks all concurrent updates to the region. Any updates that were initiated before the clear operation are allowed to complete before the region is cleared.

## Transactions with Consistent Regions

A transaction that modifies a region having consistency checking enabled generates all necessary version information for region updates when the transaction commits.

If a transaction modifies a non-replicated or empty region, the transaction is first delegated to a GemFire member that holds a replicate for the region. This behavior is similar to the transactional behavior for partitioned regions, where the partitioned region transaction is forwarded to a member that hosts the primary for the partitioned region update.

The limitation for transactions with a non-replicated or empty region is that, when consistency checking is enabled, a transaction cannot perform a `localDestroy` or `localInvalidate` operation against the region. GemFire throws an `UnsupportedOperationExceptionInTransactionException` exception in such cases. An application should use a `Destroy` or `Invalidate` operation in place of a `localDestroy` or `localInvalidate` when consistency checks are enabled.

## How Consistency Is Achieved in WAN Deployments

When two or more GemFire systems are configured to distribute events over a WAN, each system performs local consistency checking before it distributes an event to a configured gateway sender. Discarded events are not distributed across the WAN.

When a gateway sender distributes an event to another GemFire site, it adds the distributed system ID of the local cluster, as well as a timestamp for the event. In a default configuration, the cluster that receives the event examines the timestamp to determine whether or not the event should be applied. If the timestamp of the update is earlier than the local timestamp, the cluster discards the event. If the timestamp is the same as the local timestamp, then the entry having the highest distributed system ID is applied (or kept).

You can override the default consistency checking for WAN events by installing a conflict resolver plug-in for the region. If a conflict resolver is installed, then any event that can potentially cause a conflict (any event that originated from a different distributed system ID than the ID that last modified the entry) is delivered to the conflict resolver. The resolver plug-in then makes the sole determination for which update to apply or keep.

See [Resolving Conflicting Events](#) on page 200.

# Chapter 31

---

## General Region Data Management

For all regions, you have options to control memory use, back up your data to disk, and keep stale data out of your cache.

### Persistence and Overflow

You can persist data on disk for backup purposes and overflow it to disk to free up memory without completely removing the data from your cache.



**Note:** This supplements the general steps for managing data regions provided in [Basic Configuration and Programming](#) on page 131

All disk storage uses VMware® vFabric™ GemFire® [Disk Storage](#) on page 459.

#### How Persistence and Overflow Work

To use GemFire persistence and overflow, you should understand how they work with your data.

GemFire persists and overflows several types of data. You can persist or overflow the application data in your regions. In addition, GemFire persists and overflows messaging queues between distributed systems, to manage memory consumption and provide high availability.

Persistent data outlives the member where the region resides and can be used to initialize the region at creation. Overflow acts only as an extension of the region in memory.

The data is written to disk according to the configuration of GemFire disk stores. For any disk option, you can specify the name of the disk store to use or use the GemFire default disk store. See [Disk Storage](#) on page 459.

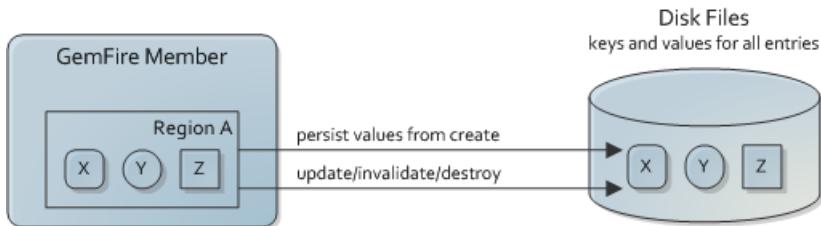
#### How Data Is Persisted and Overflowed

For persistence, the entry keys and values are copied to disk. For overflow, only the entry values are copied. Other data, such as statistics and user attributes, are retained in memory only.

- Data regions are overflowed to disk by least recently used (LRU) entries because those entries are deemed of least interest to the application and therefore less likely to be accessed.
- Server subscription queues overflow most recently used (MRU) entries. These are the messages that are at the end of the queue and so are last in line to be sent to the client.
- Multi-site gateway sender queues overflow most recently used (MRU) entries. These are the messages that are at the end of the queue and so are last in line to be sent to the remote site. You can also configure gateway sender queues to persist for high availability.

## Persistence

Persistence provides a disk backup of region entry data. The keys and values of all entries are saved to disk, like having a replica of the region on disk. Region entry operations such as put and destroy are carried out in memory and on disk.



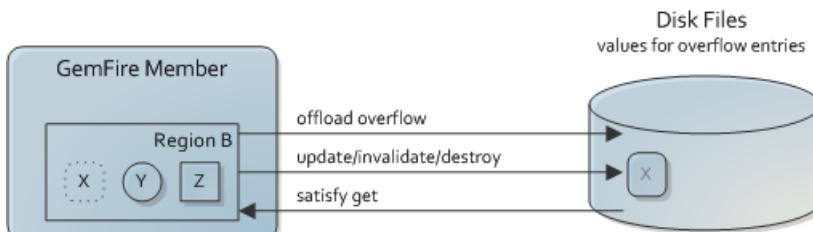
When the member stops for any reason, the region data on disk remains. In partitioned regions, where data buckets are divided among members, this can result in some data only on disk and some on disk and in memory. The disk data can be used at member startup to populate the same region.

See [How Startup and Shutdown Work with Disk Stores](#) on page 468 for information about the startup process with persistent data.

## Overflow

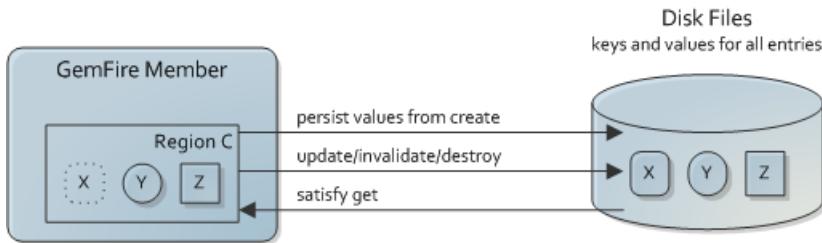
Overflow limits region size in memory by moving the values of least recently used (LRU) entries to disk. Overflow basically uses disk as a swap space for entry values. If an entry is requested whose value is only on disk, the value is copied back up into memory, possibly causing the value of a different LRU entry to be moved to disk. As with persisted entries, overflowed entries are maintained on disk just as they are in memory.

In this figure, the value of entry X has been moved to disk to make space in memory. The key for X remains in memory. From the distributed system perspective, the value on disk is as much a part of the region as the data in memory.



## Persistence and Overflow Together

Used together, persistence and overflow keep all entry keys and values on disk and only the most active entry values in memory. The removal of an entry value from memory due to overflow has no effect on the disk copy as all entries are already on disk.



## Configure Region Persistence and Overflow

Plan persistence and overflow for your data regions and configure them accordingly.

Use the following steps to configure your data regions for persistence and overflow:

1. Configure your disk stores as needed. See [Designing and Configuring Disk Stores](#) on page 463. The cache disk store defines where and how the data is written to disk.

```
<disk-store name="myPersistentStore" . . . >
<disk-store name="myOverflowStore" . . . >
```

2. Specify the persistence and overflow criteria for the region. If you are not using the default disk store, provide the disk store name in your region attributes configuration. To write asynchronously to disk, specify `disk-synchronous="false"`.

- For overflow, specify the overflow criteria in the region's `eviction-attributes` and name the disk store to use.

Example:

```
<region name="overflowRegion" . . . >
  <region-attributes disk-store-name="myOverflowStore"
disk-synchronous="true">
    <eviction-attributes>
      <!-- Overflow to disk when 100 megabytes of data reside in the
          region -->
      <lru-memory-size maximum="100" action="overflow-to-disk"/>
    </eviction-attributes>
  </region-attributes>
</region>
```

- For persistence, set the `data-policy` to `persistent-replicate` and name the disk store to use.

Example:

```
<region name="partitioned_region" refid="PARTITION_PERSISTENT">
  <region-attributes disk-store-name="myPersistentStore">
    .
  </region-attributes>
</region>
```

When you start your members, overflow and persistence will be done automatically, with the disk stores and disk write behaviors.

### Related Topics

[com.gemstone.gemfire.cache.RegionAttributes](#) for data region persistence information

[Persisting Data to Disk Example](#) on page 97

**Related Topics**

com.gemstone.gemfire.cache.EvictionAttributes for data region overflow information
<a href="#">Overflowing Data to Disk Example</a> on page 98
com.gemstone.gemfire.cache.server.ClientSubscriptionConfig
com.gemstone.gemfire.cache.util.GatewayQueueAttributes

**Overflow Configuration Examples**

The cache.xml examples show configuration of region and server subscription queue overflows.

Configure overflow criteria based on one of these factors:

- Entry count
- Absolute memory consumption
- Memory consumption as a percentage of the application heap (not available for server subscription queues)

Configuration of region overflow:

```
<!-- Overflow when the region goes over 10000 entries -->
<region-attributes>
  <eviction-attributes>
    <lru-entry-count maximum="10000" action="overflow-to-disk"/>
  </eviction-attributes>
</region-attributes>
```

Configuration of server's client subscription queue overflow:

```
<!-- Overflow the server's subscription queues when the queues reach 1 Mb
of memory -->
<cache>
  <cache-server>
    <client-subscription eviction-policy="mem" capacity="1"/>
  </cache-server>
</cache>
```

**Eviction**

Use eviction to control data region size.

**How Eviction Works**

Eviction settings cause vFabric GemFire to work to keep a region's resource use under a specified level by removing least recently used (LRU) entries to make way for new entries.

You configure for eviction based on entry count, percentage of available heap, and absolute memory usage. You also configure what to do when you need to evict: destroy entries or overflow them to disk. See [Persistence and Overflow](#) on page 245.

When GemFire determines that adding or updating an entry would take the region over the specified level, it overflows or removes enough older entries to make room. For entry count eviction, this means a one-to-one trade of an older entry for the newer one. For the memory settings, the number of older entries that need to be removed to make space depends entirely on the relative sizes of the older and newer entries.

## Eviction in Partitioned Regions

In partitioned regions, GemFire removes the oldest entry it can find *in the bucket where the new entry operation is being performed*. GemFire maintains LRU entry information on a bucket-by-bucket bases, as the cost of maintaining information across the partitioned region would be too great a performance hit.

- For memory and entry count eviction, LRU eviction is done in the bucket where the new entry operation is being performed until the overall size of the combined buckets in the member has dropped enough to perform the operation without going over the limit.
- For heap eviction, each partitioned region bucket is treated as if it were a separate region, with each eviction action only considering the LRU for the bucket, and not the partitioned region as a whole.

Because of this, eviction in partitioned regions may leave older entries for the region in other buckets in the local data store as well as in other stores in the distributed system. It may also leave entries in a primary copy that it evicts from a secondary copy or vice-versa.

## Configure Data Eviction

Use eviction controllers to configure the eviction-attributes region attribute settings to keep your region within a specified limit.

Eviction controllers monitor region and memory use and, when the limit is reached, remove older entries to make way for new data. For heap percentage, the controller used is the GemFire resource manager, configured in conjunction with the JVM's garbage collector for optimum performance.

Configure data eviction as follows. You do not need to perform these steps in the sequence shown.

1. Decide whether to evict based on:
  - Entry count (useful if your entry sizes are relatively uniform).
  - Total bytes used. In partitioned regions, this is set using `local-max-memory`. In non-partitioned, it is set in `eviction-attributes`.
  - Percentage of application heap used. This uses the GemFire resource manager. When the manager determines that eviction is required, the manager orders the eviction controller to start evicting from all regions where the eviction algorithm is set to `lru-heap-percentage`. Eviction continues until the manager calls a halt. GemFire evicts the least recently used entry hosted by the member for the region. See [Heap Use and Management](#) on page 453.
2. Decide what action to take when the limit is reached:
  - Locally destroy the entry.
  - Overflow the entry data to disk. See [Persistence and Overflow](#) on page 245.
3. Decide the maximum amount of data to allow in the member for the eviction measurement indicated. This is the maximum for all storage for the region in the member. For partitioned regions, this is the total for all buckets stored in the member for the region - including any secondary buckets used for redundancy.
4. Decide whether to program a custom sizer for your region. If you are able to provide such a class, it might be faster than the standard sizing done by GemFire. Your custom class must follow the guidelines for defining custom classes and, additionally, must implement `com.gemstone.gemfire.cache.util.ObjectSizer`. See [Requirements for Using Custom Classes in Data Caching](#) on page 159.

Examples:

```
// Create an LRU memory eviction controller with max bytes of 1000 MB
// Use a custom class for measuring the size of each object in the region
<region-attributes refid="REPLICATE">
  <eviction-attributes>
    <lru-memory-size maximum="1000" action="overflow-to-disk">
      <class-name>com.myLib.MySizer</class-name>
      <parameter name="name">
```

```

        <string>SuperSizer</string>
    </parameter>
    </lru-memory-size>
</eviction-attributes>
</region-attributes>

// Create an memory eviction controller on a partitioned region with max
bytes of 512 MB
<region name="demoPR">
    <region-attributes refid="PARTITION">
        <partition-attributes local-max-memory="512" total-num-buckets="13" />

        <eviction-attributes>
            <lru-memory-size action="local-destroy"/>
            <class-name>com.gemstone.gemfire.cache.util.ObjectSizerImpl
            </class-name>
        </eviction-attributes>
    </region-attributes>
</region>

// Configure a partitioned region for heap LRU eviction. The resource manager
controls the limits.
<region-attributes refid="PARTITION_HEAP_LRU">
</region-attributes>

AttributesFactory fac = new AttributesFactory();
fac.setEvictionAttributes(EvictionAttributes.createLRUHeapAttributes(EvictionAction.LOCAL_DESTROY));

Region currRegion = this.cache.createRegion("root",fac.create());

```

## Expiration

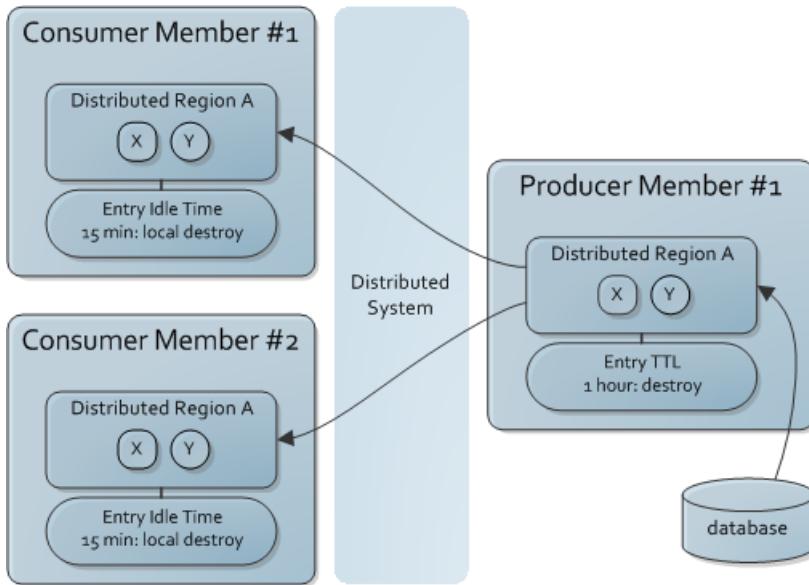
Use expiration to keep data current by removing stale entries. You can also use it to remove entries you are not using so your region uses less space. Expired entries are reloaded the next time they are requested.

### How Expiration Works

Expiration removes old entries and entries that you are not using. You can destroy or invalidate entries.

Expiration activities in distributed regions can be distributed or local. Thus, one cache could control expiration for a number of caches in the system.

This figure shows two basic expiration settings for a producer/consumer system. The producer member (on the right) populates the region from a database and the data is automatically distributed throughout the system. The data is valid only for one hour, so the producer performs a distributed destroy on entries that are an hour old. The other applications are consumers. The consumers free up space in their caches by removing their local copies of the entries for which there is no local interest (idle-time expiration). Requests for entries that have expired on the consumers will be forwarded to the producer.



## Expiration Types

vFabric GemFire uses the following expiration types:

- **Time to live (TTL).** The amount of time, in seconds, the object may remain in the cache after the last creation or update. For entries, the counter is set to zero for create and put operations. Region counters are reset when the region is created and when an entry has its counter reset. The TTL expiration attributes are `region-time-to-live` and `entry-time-to-live`.
- **Idle timeout.** The amount of time, in seconds, the object may remain in the cache after the last access. The idle timeout counter for an object is reset any time its TTL counter is reset. In addition, an entry's idle timeout counter is reset any time the entry is accessed through a get operation or a `netSearch`. The idle timeout counter for a region is reset whenever the idle timeout is reset for one of its entries. Idle timeout expiration attributes are: `region-idle-time` and `entry-idle-time`.

## Expiration Actions

vFabric GemFire uses the following expiration actions:

- `destroy`
- `local destroy`
- `invalidate` (default)
- `local invalidate`

## Partitioned Regions and Entry Expiration

For overall region performance, idle time expiration in partitioned regions may expire some entries sooner than expected. To ensure reliable read behavior across the partitioned region, we recommend that you use `entry-time-to-live` for entry expiration in partitioned regions instead of `entry-idle-time`.

Expiration in partitioned regions is executed in the primary copy, based on the primary's last accessed and last updated statistics.

- Entry updates are always done in the primary copy, resetting the primary copy's last updated and last accessed statistics.

- Entry retrieval uses the most convenient available copy of the data, which may be one of the secondary copies. This provides the best performance at the cost of possibly not updating the primary copy's statistic for last accessed time.

When the primary expires entries, it does not request last accessed statistics from the secondaries, as the performance hit would be too great. It expires entries based solely on the last time the entries were accessed in the primary copy.

You cannot use `local-destroy` or `local-invalidate` expiration actions in a partitioned region.

### Interaction Between Expiration Settings and `netSearch`

Before `netSearch` retrieves an entry value from a remote cache, it validates the *remote* entry's statistics against the *local* region's expiration settings. Entries that would have already expired in the local cache are passed over. Once validated, the entry is brought into the local cache and the local access and update statistics are updated for the local copy. The last accessed time is reset and the last modified time is updated to the time in the remote cache, with corrections made for system clock differences. Thus the local entry is assigned the true last time the entry was modified in the distributed system. The `netSearch` operation has no effect on the expiration counters in remote caches.

The `netSearch` method operates only on distributed regions with a data-policy of empty, normal and preloaded.

### Configure Data Expiration

Configure the type of expiration and the expiration action to use.

You do not have to perform these steps in the exact sequence shown.

1. Set the region's `statistics-enabled` attribute to true.



**Note:** The statistics used for expiration are available directly to the application through the `CacheStatistics` object returned by the `Region` and `Region.Entry.getStatistics` methods. The `CacheStatistics` object also provides a method for resetting the statistics counters.

2. Set the expiration attributes by expiration type, with the max times and expiration actions. See the region attributes listings for `entry-time-to-live`, `entry-idle-time`, `region-time-to-live`, and `region-idle-time` in [<region-attributes>](#) on page 882.



**Note:** For partitioned regions, to ensure reliable read behavior, use the time-to-live attributes, not the idle-time attributes. In addition, you cannot use `local-destroy` or `local-invalidate` expiration actions in partitioned regions.

Example:

```
// Setting standard expiration on an entry
<region-attributes statistics-enabled="true">
  <entry-idle-time>
    <expiration-attributes timeout="60" action="local-invalidate"/>
  </entry-idle-time>
</region-attributes>
```

3. Override the region-wide settings for specific entries, if required by your application. To do this:

- a. Program a custom expiration class that implements

`com.gemstone.gemfire.cache.CustomExpiry`. Example:

```
// Custom expiration class
// Use the key for a region entry to set entry-specific expiration
// timeouts of
// 10 seconds for even-numbered keys with a DESTROY action on the
```

```

expired entries
// Leave the default region setting for all odd-numbered keys.
public class MyClass implements CustomExpiry, Declarable
{
    private static final ExpirationAttributes CUSTOM_EXPIRY =
        new ExpirationAttributes(10, ExpirationAction.DESTROY);
    public ExpirationAttributes getExpiry(Entry entry)
    {
        int key = (Integer)entry.getKey();
        return key % 2 == 0 ? CUSTOM_EXPIRY : null;
    }
}

```

- b. Define the class inside the expiration attributes settings for the region. Example:

```

<!-- Set default entry idle timeout expiration for the region -->
<!-- Pass entries to custom expiry class for expiration overrides -->
<region-attributes statistics-enabled="true">
    <entry-idle-time>
        <expiration-attributes timeout="60" action="local-invalidate">
            <custom-expiry>
                <class-name>com.megaconglomerate.mypackage.MyClass</class-name>
                </custom-expiry>
            </expiration-attributes>
        </entry-idle-time>
    </region-attributes>

```

#### Related Topics

[Cache Expiration \(GemFire Example\)](#)

## Keeping the Cache in Sync with Outside Data Sources

Keep your distributed cache in sync with an outside data source by programming and installing application plug-ins for your region.

### Overview of Outside Data Sources

vFabric GemFire has application plug-ins to read data into the cache and write it out.

The application plug-ins:

1. Load data on cache misses using an implementation of a `com.gemstone.gemfire.cache.CacheLoader`. The `CacheLoader.load` method is called when the `get` operation can't find the value in the cache. The value returned from the loader is put into the cache and returned to the `get` operation. You might use this in conjunction with data expiration to get rid of old data, and your other data loading applications, which might be prompted by events in the outside data source. See [Configure Data Expiration](#) on page 252.
2. Write data out to the data source using the cache event handlers, `CacheWriter` and `CacheListener`. For implementation details, see [Implementing Cache Event Handlers](#) on page 288.
  - `CacheWriter` is run synchronously. Before performing any operation on a region entry, if any cache writers are defined for the region in the distributed system, the system invokes the most convenient writer. In partitioned and distributed regions, cache writers are usually defined in only a subset of the caches holding the region - often in only one cache. The cache writer can abort the region entry operation.

- `CacheListener` is run asynchronously after the cache is updated. This listener works only on local cache events, so install your listener in every cache where you want it to handle events. You can install multiple cache listeners in any of your caches.

In addition to using application plug-ins, you can also configure external JNDI database sources in your `cache.xml` and use these data sources in transactions. See [Configuring Database Connections Using JNDI](#) on page 404 for more information.

## How Data Loaders Work

By default, a region has no data loader defined. Plug an application-defined loader into any region by setting the region attribute `cache-loader` on the members that host data for the region.

The loader is called on cache misses during get operations, and it populates the cache with the new entry value in addition to returning the value to the calling thread.

A loader can be configured to load data into the GemFire cache from an outside data store. To do the reverse operation, writing data from the GemFire cache to an outside data store, use a cache writer event handler. See [Implementing Cache Event Handlers](#) on page 288.

How to install your cache loader depends on the type of region.

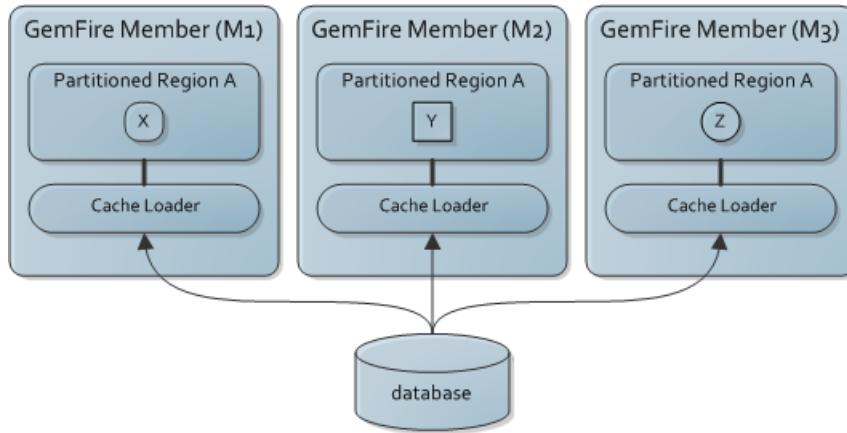
## Data Loading in Partitioned Regions

Because of the huge amounts of data they can handle, partitioned regions support partitioned loading. Each cache loader loads only the data entries in the member where the loader is defined. If data redundancy is configured, data is loaded only if the member holds the primary copy. So you must install a cache loader in every member where the partitioned attribute `local-max-memory` is not zero.

If you depend on a JDBC connection, every data store must have a connection to the data source, as shown in the following figure. Here the three members require three connections. See [Configuring Database Connections Using JNDI](#) on page 404 for information on how to configure data sources.



**Note:** Partitioned regions generally require more JDBC connections than distributed regions.

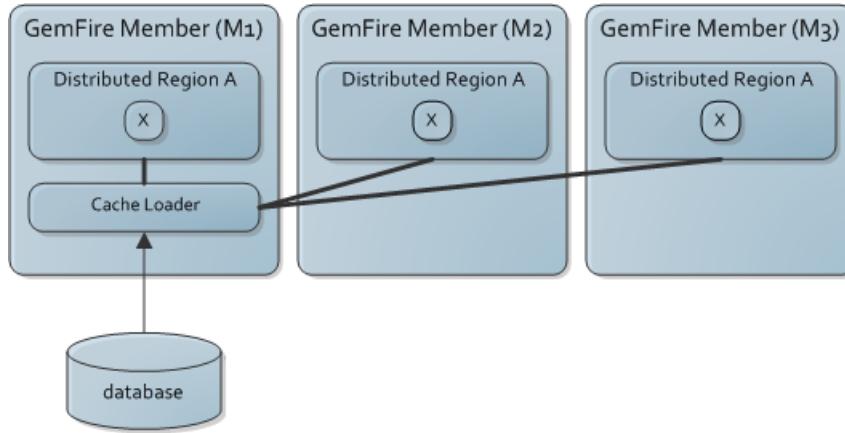


## Data Loading in Distributed Regions

In a non-partitioned distributed region, a cache loader defined in one member is available to all members that have the region defined. Loaders are usually defined in just a subset of the caches holding the region. When a

loader is needed, all available loaders for the region are invoked, starting with the most convenient loader, until the data is loaded or all loaders have been tried.

In the following figure, these members of one distributed system can be running on different machines. Loading for the distributed region is performed from M1.



### Data Loading in Local Regions

For local regions, the cache loader is available only in the member where it is defined. If a loader is defined, it is called whenever a value is not found in the local cache.

#### Implement a Data Loader

Program a data loader and configure your region to use it.

To use a data loader in your region:

1. Program your loader:
  - a. Implement `com.gemstone.gemfire.cache.CacheLoader`.
  - b. If you want to declare the loader in your `cache.xml`, implement the `com.gemstone.gemfire.cache.Declarable` interface as well.
  - c. Program the single `CacheLoader load` method to do whatever your application requires for retrieving the value from outside the cache. If you need to run Region API calls from your loader, spawn separate threads for them. Do not make direct calls to Region methods from your load method implementation as it could cause the cache loader to block, hurting the performance of the distributed system.

Example:

```
public class SimpleCacheLoader implements CacheLoader, Declarable {
    public Object load(LoaderHelper helper) {
        String key = (String) helper.getKey();
        System.out.println(" Loader called to retrieve value for " +
key);
        // Create a value using the suffix number of the key (key1,
key2, etc.)
        return "LoadedValue" + (Integer.parseInt(key.substring(3)));
    }
    public void close() { // do nothing }
    public void init(Properties props) { // do nothing }
}
```

2. Install your loader in each member region where you need it:

- In a partitioned region, install the cache loader in every data store for the region (`partition-attributes local-max-memory > 0`).
- In a distributed region, install the loader in the members where it makes sense to do so. Cache loaders are usually defined in only a subset of the members holding the region. You might, for example, assign the job of loading from a database to one or two members for a region hosted by many more members. This can be done to reduce the number of connections when the outside source is a database.

Use one of these methods to install the loader:

- XML:

```
<region-attributes>
  <cache-loader>
    <class-name>myCacheLoader</class-name>
  </cache-loader>
</region-attributes>
```

- XML with parameters:

```
<cache-loader>
  <class-name>com.company.data.DatabaseLoader</class-name>
  <parameter name="URL">
    <string>jdbc:cloudscape:rmi:MyData</string>
  </parameter>
</cache-loader>
```

- Java:

```
RegionFactory<String, Object> rf = cache.createRegionFactory(REPLICATE);
rf.setCacheLoader(new QuoteLoader());
quotes = rf.create("NASDAQ Quotes");
```

# Chapter 32

---

## Data Serialization

Data that you manage in vFabric GemFire must be serialized and deserialized for storage and transmittal between processes. You can choose among several options for data serialization.

### Overview of Data Serialization

vFabric GemFire offers serialization options other than Java serialization that give you higher performance and greater flexibility for data storage, transfers, and language types.

All data that GemFire moves out of the local cache must be serializable. However, you do not necessarily need to implement `java.io.Serializable` since other serialization options are available in GemFire. Region data that must be serializable falls under the following categories:

- Partitioned regions
- Distributed regions
- Regions that are persisted or overflowed to disk
- Server or client regions in a client/server installation
- Regions configured with a gateway sender for distributing events in a multi-site installation
- Regions that receive events from remote caches
- Regions that provide function arguments and results

To minimize the cost of serialization and deserialization, GemFire avoids changing the data format whenever possible. This means your data might be stored in the cache in serialized or deserialized form, depending on how you use it. For example, if a server acts only as a storage location for data distribution between clients, it makes sense to leave the data in serialized form, ready to be transmitted to clients that request it. Partitioned region data is always stored in serialized form with one exception-- functions that add data to a partitioned region locally use the deserialized form.

### Data Serialization Options

With GemFire, you have the option to serialize your domain objects automatically or to implement serialization using one of GemFire's interfaces. Enabling automatic serialization means that domain objects are serialized and deserialized without your having to make any code changes to those objects. This automatic serialization is performed by registering your domain objects with a custom `PdxSerializer` called the `ReflectionBasedAutoSerializer`, which uses Java reflection to infer which fields to serialize.

If autoserialization does not meet your needs, you can serialize your objects by implementing one of the GemFire interfaces, `PdxSerializable` or `DataSerializable`. You can use these interfaces to replace any standard Java data serialization for better performance. If you cannot or do not want to modify your domain classes, each interface has an alternate serializer class, `PdxSerializer` and `DataSerializer`. To use these, you create your custom serializer class and then associate it with your domain class in the GemFire cache configuration.

GemFire Data serialization is about 25% faster than PDX serialization, however using PDX serialization will help you to avoid the even larger costs of performing deserialization.

**Table 50: Serialization Options: Comparison of Features**

Capability	GemFire Data Serializable	GemFire PDX Serializable
Implements Java Serializable.	X	
Handles multiple versions of application domain objects, providing the versions differ by the addition or subtraction of fields.		X
Provides single field access of serialized data, without full deserialization - supported also for OQL querying.		X
Automatically ported to other languages by GemFire		X
Works with .NET clients.	X	X
Works with C++ clients.	X	X
Works with GemFire delta propagation.	X	X (See note below.)



**Note:** By default, you can use GemFire delta propagation with PDX serialization. However, delta propagation will not work if you have set the GemFire property `read-serialized` to "true". In terms of deserialization, to apply a change delta propagation requires a domain class instance and the `fromDelta` method. If you have set `read-serialized` to true, then you will receive a `PdxInstance` instead of a domain class instance and `PdxInstance` does not have the `fromDelta` method required for delta propagation.

### Differences between GemFire Serialization (PDX or Data Serializable) and Java Serialization

GemFire serialization (either PDX serialization or data Serializable) does not support circular object graphs whereas Java serialization does. In GemFire serialization, if the same object is referenced more than once in an object graph, the object is serialized for each reference, and deserialization produces multiple copies of the object. By contrast in this situation, Java serialization serializes the object once and when deserializing the object, it produces one instance of the object with multiple references.

## vFabric GemFire PDX Serialization

vFabric GemFire's Portable Data eXchange (PDX) is a cross-language data format that can reduce the cost of distributing and serializing your objects. PDX stores data in named fields that you can access individually, to avoid the cost of deserializing the entire data object. PDX also allows you to mix versions of objects where you have added or removed fields.

### GemFire PDX Serialization Features

GemFire PDX serialization offers several advantages in terms of functionality.

### Application Versioning of PDX Domain Objects

Domain objects evolve along with your application code. You might create an address object with two address lines, then realize later that a third line is required for some situations. Or you might realize that a particular field is not used and want to get rid of it. With PDX, you can use old and new versions of domain objects together in a distributed system if the versions differ by the addition or removal of fields. This compatibility lets you gradually introduce modified code and data into the system, without bringing the system down.

GemFire maintains a central registry of the PDX domain object metadata. Using the registry, GemFire preserves fields in each member's cache regardless of whether the field is defined. When a member receives an object with a registered field that the member is not aware of, the member does not access the field, but preserves it and passes it along with the entire object to other members. When a member receives an object that is missing one or more fields according to the member's version, GemFire assigns the Java default values for the field types to the missing fields.

### **Portability of PDX Serializable Objects**

When you serialize an object using PDX, GemFire stores the object's type information in the central registry. The information is passed among clients and servers, peers, and distributed systems.

This centralization of object type information is advantageous for client/server installations in which clients and servers are written in different languages. Clients pass registry information to servers automatically when they store a PDX serialized object. Clients can run queries and functions against the data in the servers without compatibility between server and the stored objects. One client can store data on the server to be retrieved by another client, with no requirements on the part of the server. Thus you can code .NET clients to manage data using Java servers without having to create Java implementations of your .NET domain objects.

### **Reduced Deserialization of Serialized Objects**

The access methods of PDX serialized objects allow you to examine specific fields of your domain object without deserializing the entire object. Depending on your object usage, you can reduce serialization and deserialization costs significantly.

Java and .NET clients can run queries and execute functions against the objects in the server caches without deserializing the entire object on the server side. The query engine automatically recognizes PDX objects, retrieves the `PdxInstance` of the object and uses only the fields it needs. Likewise, peers can access only the necessary fields from the serialized object, keeping the object stored in the cache in serialized form.

### **High Level Steps for Using PDX Serialization**

To use PDX serialization, you can configure and use vFabric GemFire's reflection-based autoserializer, or you can program the serialization of your objects by using the PDX interfaces and classes.

Optionally, program your application code to deserialize individual fields out of PDX representations of your serialized objects. You may also need to persist your PDX metadata to disk for recovery on startup.

### **Procedure**

1. Use one of these serialization options for each object type that you want to serialize using PDX serialization:
  - [Using Automatic Reflection-Based PDX Serialization](#) on page 260
  - [Serializing Your Domain Object with a PdxSerializer](#) on page 265
  - [Implementing PdxSerializable in Your Domain Object](#) on page 267
2. To ensure that your servers do not need to load the application classes, set the `<pdx> read-serialized` attribute to true in the server's `cache.xml` file.
3. If you are storing any GemFire data on disk, then you must configure PDX serialization to use persistence. See [Persisting PDX Metadata to Disk](#) on page 272 for more information.
4. For multisite (WAN) installations only. If you will use PDX serialization in any of your WAN-enabled regions, for each distributed system, you must choose a unique integer between 0 (zero) and 255 and set the `distributed-system-id` in every member's `gemfire.properties` file. See [Configuring a Multi-site \(WAN\) System](#) on page 195.
5. (Optional) Wherever you run explicit application code to retrieve and manage your cached entries, you may want to manage your data objects without using full deserialization. To do this, see [Programming Your Application to Use PdxInstances](#) on page 268.

## Using Automatic Reflection-Based PDX Serialization

You can configure your cache to automatically serialize and deserialize domain objects without having to add any extra code to them.

You can automatically serialize and deserialize domain objects without making any code changes to those objects and without coding a `PdxSerializer` class. You do this by registering your domain objects with a custom `PdxSerializer` called `ReflectionBasedAutoSerializer` that uses Java reflection to infer which fields to serialize.

You can also extend the `ReflectionBasedAutoSerializer` to customize its behavior. For example, you could add optimized serialization support for `BigInteger` and `BigDecimal` types. See [Extending the `ReflectionBasedAutoSerializer`](#) on page 262 for details.



**Note:** Your custom PDX autoserializable classes cannot use the `com.gemstone` package. If they do, the classes will be ignored by the PDX auto serializer.

### Prerequisites

- Understand generally how to configure the GemFire cache.
- Understand how PDX serialization works and how to configure your application to use `PdxSerializer`.

### Procedure

In your application where you manage data from the cache, provide the following configuration and code as appropriate:

1. Using one of the following methods, set the PDX serializer to `ReflectionBasedAutoSerializer`.

- a. In `cache.xml`:

```
<!-- Cache configuration configuring auto serialization behavior -->
<cache>
  <pdx>
    <pdx-serializer>
      <class-name>
        com.gemstone.gemfire.pdx.ReflectionBasedAutoSerializer
      </class-name>
      <parameter name="classes">
        <string>com.company.domain.DomainObject</string>
      </parameter>
    </pdx-serializer>
  </pdx>
  ...
</cache>
```

The parameter, `classes`, takes a comma-separated list of class patterns to define the domain classes to serialize. If your domain object is an aggregation of other domain classes, you need to register the domain object and each of those domain classes explicitly for the domain object to be serialized completely.

- b. Using the Java API:

```
Cache c = new CacheFactory()
  .setPdxSerializer(new
ReflectionBasedAutoSerializer("com.company.domain.DomainObject" ))
  .create();
```

2. Customize the behavior of the `ReflectionBasedAutoSerializer` using one of the following mechanisms:

- By using a class pattern string to specify the classes to auto-serialize and customize how the classes are serialized. Class pattern strings can be specified in the API by passing strings to the `ReflectionBasedAutoSerializer` constructor or by specifying them in `cache.xml`. See [Customizing Serialization with Class Pattern Strings](#) on page 261 for details.
  - By creating a subclass of `ReflectionBasedAutoSerializer` and overriding specific methods. See [Extending the ReflectionBasedAutoSerializer](#) on page 262 for details.
3. If desired, configure the `ReflectionBasedAutoSerializer` to check the portability of the objects it is passed before it tries to autoserialize them. When this flag is set to true, the `ReflectionBasedAutoSerializer` will throw a `NonPortableClassException` error when trying to autoserialize a non-portable object. To set this, use the following configuration:

- In `cache.xml`:

```
<!-- Cache configuration configuring auto serialization behavior -->
<cache>
  <pdx>
    <pdx-serializer>
      <class-name>
        com.gemstone.gemfire.pdx.ReflectionBasedAutoSerializer
      </class-name>
      <parameter name="classes">
        <string>com.company.domain.DomainObject</string>
      </parameter>
      <parameter name="check-portability">
        <string>true</string>
      </parameter>
    </pdx-serializer>
  </pdx>
  ...
</cache>
```

- Using the Java API:

```
Cache c = new CacheFactory()
  .setPdxSerializer(new
    ReflectionBasedAutoSerializer(true, "com.company.domain.DomainObject"))
  .create();
```

For each domain class you provide, all fields are considered for serialization except those defined as `static` or `transient` and those you explicitly exclude using the class pattern strings.



**Note:** The `ReflectionBasedAutoSerializer` traverses the given domain object's class hierarchy to retrieve all fields to be considered for serialization. So if `DomainObjectB` inherits from `DomainObjectA`, you only need to register `DomainObjectB` to have all of `DomainObjectB` serialized.

## Customizing Serialization with Class Pattern Strings

Use class pattern strings to name the classes that you want to serialize using GemFire's reflection-based autoserializer and to specify object identity fields and to specify fields to exclude from serialization.

The class pattern strings used to configured the `ReflectionBasedAutoSerializer` are standard regular expressions. For example, this expression would select all classes defined in the `com.company.domain` package and its subpackages:

```
com\.\company\.\domain\..*
```

You can augment the pattern strings with a special notation to define fields to exclude from serialization and to define fields to mark as PDX identity fields. The full syntax of the pattern string is:

```
<class pattern> [# (identity|exclude) = <field pattern>]... [, <class pattern>...]
```

The following example pattern string sets these PDX serialization criteria:

- Classes with names matching the pattern `com.company.DomainObject.*` are serialized. In those classes, fields beginning with `id` are marked as identity fields and fields named `creationDate` are not serialized.
- The class `com.company.special.Patient` is serialized. In the class, the field, `ssn` is marked as an identity field

```
com.company.DomainObject.*#identity=id.*#exclude=creationDate,  
com.company.special.Patient#identity=ssn
```



**Note:** There is no association between the `identity` and `exclude` options, so the pattern above could also be expressed as:

```
com.company.DomainObject.*#identity=id.*,  
com.company.DomainObject.*#exclude=creationDate,  
com.company.special.Patient#identity=ssn
```



**Note:** The order of the patterns is not relevant. All defined class patterns are used when determining whether a field should be considered as an identity field or should be excluded.

Examples:

- This XML uses the example pattern shown above:

```
<parameter name="classes">  
    <string>com.company.DomainObject.*#identity=id.*#exclude=creationDate,  
    com.company.special.Patient#identity=ssn</string>  
</parameter>
```

- This application code sets the same pattern:

```
classPatterns.add("com.company.DomainObject.*#identity=id.*#exclude=creationDate,  
    com.company.special.Patient#identity=ssn");
```

- This application code has the same effect:

```
Cache c = new CacheFactory().set("cache-xml-file", cacheXmlFileName)  
    .setPdxSerializer(new  
ReflectionBasedAutoSerializer("com.foo.DomainObject#identity=id.*",  
    "com.company.DomainObject.*#exclude=creationDate", "com.company.special.Patient#identity=ssn"))  
    .create();
```

### Extending the ReflectionBasedAutoSerializer

You can extend the `ReflectionBasedAutoSerializer` to handle serialization in a customized manner. This section provides an overview of the available method-based customization options and an example of extending the serializer to support `BigDecimal` and `BigInteger` types.

### Reasons to Extend the ReflectionBasedAutoSerializer

One of the main use cases for extending the `ReflectionBasedAutoSerializer` is that you want it to handle an object that would currently need to be handled by standard Java serialization. There are several issues

with having to use standard Java serialization that can be addressed by extending the PDX `ReflectionBasedAutoSerializer`.

- Each time we transition from a GemFire serialized object to an object that will be Java I/O serialized, extra data must get serialized. This can cause a great deal of serialization overhead. This is why it is worth extending the `ReflectionBasedAutoSerializer` to handle any classes that normally would have to be Java I/O serialized.
- Expanding the number of classes that can use the `ReflectionBasedAutoSerializer` is beneficial when you encounter object graphs. After we use Java I/O serialization on an object, any objects under that object in the object graph will also have to be Java I/O serialized. This includes objects that normally would have been serialized using PDX or `DataSerializable`.
- If standard Java I/O serialization is done on an object and you have enabled check-portability, then an exception will be thrown. Even if you are not concerned with the object's portability to .NET, you can use this flag to find out what classes would use standard Java serialization (by getting an exception on them) and then enhancing your auto serializer to handle them.

### Overriding `ReflectionBasedAutoSerializer` Behavior

You can customize the specific behaviors in `ReflectionBasedAutoSerializer` by overriding the following methods:

- `isClassAutoSerialized` customizes which classes to autoserialize.
- `isFieldIncluded` specifies which fields of a class to autoserialize.
- `getFieldName` defines the specific field names that will be generated during autoserialization.
- `isIdentifyField` controls which field is marked as the identity field. Identity fields are used when a `PdxInstance` computes its hash code to determine whether it is equal to another object.
- `getFieldType` determines the field type that will be used when autoserializing the given field.
- `transformFieldValue` controls whether specific field values of a PDX object can be transformed during serialization.
- `writeTransform` controls what field value is written during auto serialization.
- `readTransform` controls what field value is read during auto deserialization.

These methods are only called the first time the `ReflectionBasedAutoSerializer` sees a new class. The results will be remembered and used the next time the same class is seen.

For details on these methods and their default behaviors, see [ReflectionBasedAutoSerializer JavaDocs](#) for details.

### Example of Optimizing Auterialization of `BigInteger` and `BigDecimal` Types

This section provides an example of extending the `ReflectionBasedAutoSerializer` to optimize the automatic serialization of `BigInteger` and `BigDecimal` types.

The following code sample illustrates a subclass of the `ReflectionBasedAutoSerializer` that optimizes `BigInteger` and `BigDecimal` auterialization:

```
public static class BigAutoSerializer extends ReflectionBasedAutoSerializer {
    public BigAutoSerializer(Boolean checkPortability, string... patterns) {
        super(checkPortability, patterns);
    }

    @Override
    public FieldType get FieldType(Field f, Class<?> clazz) {
        if (f.getType().equals(BigInteger.class)) {
            return FieldType.BYTE_ARRAY;
        } else if (f.getType().equals(BigDecimal.class)) {
            return FieldType.STRING;
        } else {
    }
}
```

```
        return super.getFieldType(f, clazz);
    }
}
@Override
public boolean transformFieldValue(Field f, Class<?> clazz) {
    if (f.getType().equals(BigInteger.class)) {
        return true;
    } else if (f.getType().equals(BigDecimal.class)) {
        return true;
    } else {
        return super.transformFieldValue(f, clazz);
    }
}

@Override
public Object writeTransform(Field f, Class<?> clazz, Object
originalValue) {
    if (f.getType().equals(BigInteger.class)) {
        byte[] result = null;
        if (originalValue != null) {
            BigInteger bi = (BigInteger)originalValue;
            result = bi.toByteArray();
        }
        return result;
    } else if (f.getType().equals(BigDecimal.class)) {
        Object result = null;
        if (originalValue != null) {
            BigDecimal bd = (BigDecimal)originalValue;
            result = bd.toString();
        }
        return result;
    } else {
        return super.writeTransform(f, clazz, originalValue);
    }
}

@Override
public Object readTransform(Field f, Class<?> clazz, Object
serializedValue) {
    if (f.getType().equals(BigInteger.class)) {
        BigInteger result = null;
        if (serializedValue != null) {
            result = new BigInteger((byte[])serializedValue);
        }
        return result;
    } else if (f.getType().equals(BigDecimal.class)) {
        BigDecimal result = null;
        if (serializedValue != null) {
            result = new BigDecimal((String)serializedValue);
        }
        return result;
    } else {
        return super.readTransform(f, clazz, serializedValue);
    }
}
```

```
}
```

### Serializing Your Domain Object with a PdxSerializer

For a domain object that you cannot or do not want to modify, use the `PdxSerializer` class to serialize and deserialize the object's fields. You use one `PdxSerializer` implementation for the entire cache, programming it for all of the domain objects that you handle in this way.

With `PdxSerializer`, you leave your domain object as-is and handle the serialization and deserialization in the separate serializer. You register the serializer in your cache PDX configuration. Program the serializer to handle all of the domain objects you need.

If you write your own `PdxSerializer` and you also use the `ReflectionBasedAutoSerializer`, then the `PdxSerializer` needs to own the `ReflectionBasedAutoSerializer` and delegate to it. A Cache can only have a single `PdxSerializer` instance.



**Note:** The `PdxSerializer` `toData` and `fromData` methods differ from those for `PdxSerializable`. They have different parameters and results.

### Procedure

1. If you have not already implemented `PdxSerializer` for some other domain object, perform these steps:
  - a. Create a new class as your cache-wide serializer and make it implement `PdxSerializer`. If you want to declare your new class in the `cache.xml` file, have it also implement `Declarable`.

Example:

```
import com.gemstone.gemfire.cache.Declarable;
import com.gemstone.gemfire.pdx.PdxReader;
import com.gemstone.gemfire.pdx.PdxSerializer;
import com.gemstone.gemfire.pdx.PdxWriter;

public class ExamplePdxSerializer implements PdxSerializer, Declarable
{
  ...
}
```

- b. In your cache pdx configuration, register the serializer class in the cache's `<pdx> <pdx-serializer> <class-name>` attribute.

Example:

```
// Configuration setting PDX serializer for the cache
<cache>
  <pdx>
    <pdx-serializer>
      <class-name>com.company.ExamplePdxSerializer</class-name>
    </pdx-serializer>
  </pdx>
  ...
</cache>
```

Or use the `CacheFactory.setPdxSerializer` API.

```
Cache c = new CacheFactory
  .setPdxSerializer(new ExamplePdxSerializer())
  .create();
```

2. Program `PdxSerializer.toData` to recognize, cast, and handle your domain object:

- a. Write each standard Java data field of your domain class using the `PdxWriter` write methods.
- b. Call the `PdxWriter markIdentityField` method for each field you want to have GemFire use to identify your object. Put this after the field's write method. GemFire uses this information to compare objects for operations like distinct queries. If you do not set at least one identity field, then the `equals` and `hashCode` methods will use all PDX fields to compare objects and consequently, will not perform as well. It is important that the fields used by your `equals` and `hashCode` implementations are the same fields that you mark as identity fields.
- c. For a particular version of your class, you need to consistently write the same named field each time. The field names or number of fields must not change from one instance to another for the same class version.
- d. For best performance, do fixed width fields first and then variable length fields.
- e. If desired, you can check the portability of the object before serializing it by adding the `checkPortability` parameter when using the `PdxWriter.writeObject`, `writeObjectArray`, and `writeField` methods.

Example `toData` code:

```
public boolean toData(Object o, PdxWriter writer)
{
    if(!(o instanceof PortfolioPdx)) {
        return false;
    }

    PortfolioPdx instance = (PortfolioPdx) o;
    writer.writeInt("id", instance.id)
    //identity field
    .markIdentityField("id")
    .writeDate("creationDate", instance.creationDate)
    .writeString("pkid", instance.pkid)
    .writeObject("positions", instance.positions)
    .writeString("type", instance.type)
    .writeString("status", instance.status)
    .writeStringArray("names", instance.names)
    .writeByteArray("newVal", instance.newVal)

    return true;
}
```

- a. Program `PdxSerializer.fromData` to create an instance of your class, read your data fields from the serialized form into the object's fields using the `PdxReader` read methods, and return the created object.

Provide the same names that you did in `toData` and call the read operations in the same order as you called the write operations in your `toData` implementation.

GemFire provides the domain class type and `PdxReader` to the `fromData` method.

Example `fromData` code:

```
public Object fromData(Class<?> clazz, PdxReader reader)
{
    if(!clazz.equals(PortfolioPdx.class)) {
        return null;
    }

    PortfolioPdx instance = new PortfolioPdx();
    instance.id = reader.readInt("id");
    instance.creationDate = reader.readDate("creationDate");
    instance.pkid = reader.readString("pkid");
    instance.positions = (Map<String,
PositionPdx>)reader.readObject("positions");
    instance.type = reader.readString("type");
```

```

        instance.status = reader.readString("status");
        instance.names = reader.readStringArray("names");
        instance.newVal = reader.readByteArray("newVal");

        return instance;
    }
}

```

3. If desired, you can also enable extra validation in your use of `PdxWriter`. You can set this by setting the system property `gemfire.validatePdxWriters` to `true`. Note that you should only set this option if you are debugging new code as this option can decrease system performance.

### Implementing `PdxSerializable` in Your Domain Object

For a domain object with source that you can modify, implement the `PdxSerializable` interface in the object and use its methods to serialize and deserialize the object's fields.

#### Procedure

1. In your domain class, implement `PdxSerializable`, importing the required `com.gemstone.gemfire.pdx` classes.

For example:

```

import com.gemstone.gemfire.pdx.PdxReader;
import com.gemstone.gemfire.pdx.PdxSerializable;
import com.gemstone.gemfire.pdx.PdxWriter;

public class PortfolioPdx implements PdxSerializable {
    ...
}

```

2. If your domain class does not have a zero-arg constructor, create one for it.

For example:

```

public PortfolioPdx(){
}

```

3. Program `PdxSerializable.toData`.

- a. Write each standard Java data field of your domain class using the `PdxWriter` write methods. GemFire automatically provides `PdxWriter` to the `toData` method for `PdxSerializable` objects.
- b. Call the `PdxWriter markIdentifyField` method for each field you want to have GemFire use to identify your object. Put this after the field's write method. GemFire uses this information to compare objects for operations like distinct queries. If you do not set at least one identity field, then the `equals` and `hashCode` methods will use all PDX fields to compare objects and consequently, will not perform as well. It is important that the fields used by your `equals` and `hashCode` implementations are the same fields that you mark as identity fields.
- c. For a particular version of your class, you need to consistently write the same named field each time. The field names or number of fields must not change from one instance to another for the same class version.
- d. For best performance, do fixed width fields first and then variable length fields.

Example `toData` code:

```

// PortfolioPdx fields
private int id;
private String pkid;
private Map<String, PositionPdx> positions;
private String type;
private String status;
private String[] names;
private byte[] newVal;

```

```

private Date creationDate;
...

public void toData(PdxWriter writer)
{
    writer.writeInt("id", id)
    // The markIdentityField call for a field must
    // come after the field's write method
    .markIdentityField("id")
    .writeDate("creationDate", creationDate) //fixed length field
    .writeString("pkid", pkid)
    .writeObject("positions", positions)
    .writeString("type", type)
    .writeString("status", status)
    .writeStringArray("names", names)
    .writeByteArray("newVal", newVal)
}

```

4. Program `PdxSerializable.fromData` to read your data fields from the serialized form into the object's fields using the `PdxReader` read methods.

Provide the same names that you did in `toData` and call the read operations in the same order as you called the write operations in your `toData` implementation.

GemFire automatically provides `PdxReader` to the `fromData` method for `PdxSerializable` objects.

Example `fromData` code:

```

public void fromData(PdxReader reader)
{
    id = reader.readInt("id");
    creationDate = reader.readDate("creationDate");
    pkid = reader.readString("pkid");
    position1 = (PositionPdx)reader.readObject("position1");
    position2 = (PositionPdx)reader.readObject("position2");
    positions = (Map<String, PositionPdx>)reader.readObject("positions");

    type = reader.readString("type");
    status = reader.readString("status");
    names = reader.readStringArray("names");
    newVal = reader.readByteArray("newVal");
    arrayNull = reader.readByteArray("arrayNull");
    arrayZeroSize = reader.readByteArray("arrayZeroSize");
}

```

### What to do next

- As needed, configure and program your GemFire applications to use `PdxInstance` for selective object deserialization. See [Programming Your Application to Use PdxInstances](#) on page 268.

### **Programming Your Application to Use PdxInstances**

A `PdxInstance` is a light-weight wrapper around PDX serialized bytes. It provides applications with run-time access to fields of a PDX serialized object.

You can configure your cache to return a `PdxInstance` when a PDX serialized object is deserialized instead of deserializing the object to a domain class. You can then program your application code that reads your entries to handle `PdxInstances` fetched from the cache.



**Note:** This applies only to entry retrieval that you explicitly code using methods like `EntryEvent.getNewValue` and `Region.get`, as you do inside functions or in cache listener code. This does not apply to querying because the query engine retrieves the entries and handles object access for you.

If you configure your cache to allow PDX serialized reads, a fetch from the cache returns the data in the form it is found. If the object is not serialized, the fetch returns the domain object. If the object is serialized, the fetch returns the `PdxInstance` for the object.



**Note:** If you are using `PdxInstances`, you cannot use delta propagation to apply changes to PDX serialized objects.

For example, in client/server applications that are programmed and configured to handle all data activity from the client, PDX serialized reads done on the server side will always return a `PdxInstance`. This is because all of data is serialized for transfer from the client, and you are not performing any server-side activities that would deserialize the objects in the server cache.

In mixed situations, such as where a server cache is populated from client operations and also from data loads done on the server side, fetches done on the server can return a mix of `PdxInstances` and domain objects.

When fetching data in a cache with PDX serialized reads enabled, the safest approach is to code to handle both types, receiving an `Object` from the fetch operation, checking the type and casting as appropriate. However, if you know that the class is not available in the JVM, then you can avoid performing the type check.



**Note:** `PdxInstance` overrides any custom implementation you might have coded for your object's `equals` and `hashCode` methods. Make sure you have marked at least one identity field when writing PDX serialized objects. If you do not set at least one identity field, then the `PdxInstance.equals` and `hashCode` methods will use all PDX fields to compare objects and consequently, will not perform as well.

## Prerequisites

- Understand generally how to configure the GemFire cache. See [Basic Configuration and Programming](#) on page 131.

## Procedure

In your application where you fetch data from the cache, provide the following configuration and code as appropriate:

- In the `cache.xml` file of the member where entry fetches are run, set the `<pdx> read-serialized` attribute to true.



**Note:** Data is not necessarily accessed on the member that you have coded for it. For example, if a client application runs a function on a server, the actual data access is done on the server, so you set `read-serialized` to true on the server.

For example:

```
// Cache configuration setting PDX read behavior
<cache>
  <pdx read-serialized="true" />
  ...
</cache>
```

- Write the application code that fetches data from the cache to handle a `PdxInstance`. If you are sure you will only retrieve `PdxInstances` from the cache, you can code only for that. In many cases, a

`PdxInstance` or a domain object may be returned from your cache entry retrieval operation, so you should check the object type and handle each possible type.

For example:

```
// put/get code with serialized read behavior
// put is done as normal
myRegion.put(myKey, myPdxSerializableObject);

// get checks Object type and handles each appropriately
Object myObject = myRegion.get(myKey);
if (myObject instanceof PdxInstance) {
    // get returned PdxInstance instead of domain object
    PdxInstance myPdxInstance = (PdxInstance)myObject;

    // PdxInstance.getField deserializes the field, but not the object
    String fieldValue = myPdxInstance.getField("stringFieldName");

    // Update a field and put it back into the cache
    // without deserializing the entire object
    WritablePdxInstance myWritablePdxI = myPdxInstance.createWriter();
    myWritablePdxI.setField("fieldName", fieldValue);
    region.put(key, myWritablePdxI);

    // Deserialize the entire object if needed, from the PdxInstance
    DomainClass myPdxObject = (DomainClass);
}
else if (myObject instanceof DomainClass) {
    // get returned instance of domain object
    // code to handle domain object instance
    ...
}
...
...
```



**Note:** Due to a limitation with PDX, if your PDX-enabled cache contains `TreeSet` domain objects, you should implement a Comparator that can handle both your domain objects and `PdxInstance` objects. You will also need to make the domain classes available on the server.

### Adding JSON Documents to the GemFire Cache

The `JSONFormatter` API allows you to put JSON formatted documents into regions and retrieve them later by storing the documents internally as `PdxInstances`.

vFabric GemFire now supports the use of JSON formatted documents natively. When you add a JSON document to a GemFire cache, you call the `JSONFormatter` APIs to transform them into the PDX format (as a `PdxInstance`), which enables GemFire to understand the JSON document at a field level.

In terms of querying and indexing, because the documents are stored internally as PDX, applications can index on any field contained inside the JSON document including any nested field (within JSON objects or JSON arrays.) Any queries run on these stored documents will return `PdxInstances` as results. To update a JSON document stored in GemFire, you can execute a function on the `PdxInstance`.

You can then use the `JSONFormatter` to convert the `PdxInstance` results back into the JSON document.

`JSONFormatter` uses a streaming parser ([Jackson](#), JSON processor) to turn JSON documents into the optimized PDX format. You must have added the `jackson-core-asl-1.9.9.jar` included in `$GEMFIRE/lib` to your CLASSPATH.

The `JSONFormatter` class has four static methods that are used to convert JSON document into `PdxInstances` and then to convert those `PdxInstances` back into JSON document.

You need to call the following methods before putting any JSON document into the GemFire region:

- `fromJSON`. Creates a PdxInstance from a JSON byte array. Returns the PdxInstance.
- `fromJSON`. Creates a PdxInstance from a JSON string. Returns the PdxInstance.

After putting the JSON document into a region as a PdxInstance, you can execute standard GemFire queries and create indexes on the JSON document in the same manner you would query or index any other GemFire PdxInstance.

After executing a GemFire query or calling `region.get`, you can use the following methods to convert a PdxInstance back into the JSON format .

- `toJSON`. Reads a PdxInstance and returns a JSON string.
- `toJSONArray`. Reads a PdxInstance and returns a JSON byte array.

For more information on using the `JSONFormatter`, see the Java API documentation for `com.gemstone.gemfire.pdx.JSONFormatter`.

### **Using PdxInstanceFactory to Create PdxInstances**

You can use the `PdxInstanceFactory` interface to create a `PdxInstance` from raw data when the domain class is not available on the server.

This can be particularly useful when you need an instance of a domain class for plug in code such as a function or a loader. If you have the raw data for the domain object (the class name and each field's type and data), then you can explicitly create a `PdxInstance`. The `PdxInstanceFactory` is very similar to the `PdxWriter` except that after writing each field, you need to call the `create` method which returns the created `PdxInstance`.

To create a factory call `RegionService.createPdxInstanceFactory`. A factory can only create a single instance. To create multiple instances create multiple factories or use `PdxInstance.createWriter()` to create subsequent instances. Using `PdxInstance.createWriter()` is usually faster.

When you create a `PdxInstance`, set at least one identity field using the `markIdentityField` method. If you do not mark an identity field, the `PdxInstance.equals` and `hashCode` methods will use all PDX fields to compare objects and consequently, will not perform as well. It is important that the fields used by your `equals` and `hashCode` implementations are the same fields that you mark as identity fields.

The following is a code example of using `PdxInstanceFactory`:

```
PdxInstance pi = cache.createPdxInstanceFactory( "com.company.DomainObject" )
    .writeInt( "id", 37 )
    .markIdentityField( "id" )
    .writeString( "name", "Mike Smith" )
    .writeObject( "favoriteDay", cache.createPdxEnum( "com.company.Day" ,
    "FRIDAY", 5 ) )
    .create();
```

For more information, see `PdxInstanceFactory` in the Java API documentation.

### **Enum Objects as PdxInstances**

You can now work with enum objects as `PdxInstances`. When you fetch an enum object from the cache, you can now deserialize it as a `PdxInstance`. To check whether a `PdxInstance` is an enum, use the `PdxInstance.isEnum` method. A enum `PdxInstance` will have one field named "name" whose value is a `String` that corresponds to the enum constant name.

An enum `PdxInstance` is not writable; if you call `createWriter` it will throw an exception.

The `RegionService` has a new method that allows you to create a `PdxInstance` that represents an enum. See `RegionService.createPdxEnum` in the Java API documentation.

## Persisting PDX Metadata to Disk

GemFire allows you to persist PDX metadata to disk and specify the disk store to use.

### Prerequisites

- Understand generally how to configure the GemFire cache. See [Basic Configuration and Programming](#) on page 131.
- Understand how GemFire disk stores work. See [Disk Storage](#) on page 459.

### Procedure

1. Set the <pdx> attribute `persistent` to true in your cache configuration. This is required for caches that use PDX with persistent regions and with regions that use a gateway sender to distribute events across a WAN. Otherwise, it is optional.
2. (Optional) If you want to use a disk store that is not the GemFire default disk store, set the <pdx> attribute `disk-store-name` to the name of your non-default disk store.



**Note:** If you are using PDX serialized objects as region entry keys and you are using persistent regions, then you must configure your PDX disk store to be a different one than the disk store used by the persistent regions.

This example `cache.xml` enables PDX persistence and sets a non-default disk store in a server cache configuration:

```
<pdx read-serialized="true"
      persistent="true" disk-store-name="SerializationDiskStore">
  <pdx-serializer>
    <class-name>pdxSerialization.defaultSerializer</class-name>
  </pdx-serializer>
</pdx>
<region ...>
```

#### Related Topics

[Configure Region Persistence and Overflow](#) on page 247

[Create Data Regions for Multi-site Communication](#) on page 197

## Using PDX Objects as Region Entry Keys

Using PDX objects as region entry keys is highly discouraged.

The best practice for creating region entry keys is to use a simple key; for example, use a String or Integer. If the key must be a domain class, then you should use a non-PDX-serialized class.

If you must use PDX serialized objects as region entry keys, ensure that you do not set `read-serialized` to true. This configuration setting will cause problems in partitioned regions because partitioned regions require the hash code of the key to be the same on all JVMs in the distributed system. When the key is a `PdxInstance` object, its hash code will likely not be the same as the hash code of the domain object.

If you are using PDX serialized objects as region entry keys and you are using persistent regions, then you must configure your PDX disk store to be a different one than the disk store used by the persistent regions. See [Persisting PDX Metadata to Disk](#) on page 272 for information on setting up a PDX disk store.

## vFabric GemFire Data Serialization (DataSerializable and DataSerializer)

vFabric GemFire's `DataSerializable` interface gives you quick serialization of your objects.

## Data Serialization with the DataSerializable Interface

GemFire's `DataSerializable` interface gives you faster and more compact data serialization than the standard Java serialization or GemFire PDX serialization. However, while GemFire `DataSerializable` interface is generally more performant than GemFire's `PdxSerializable`, it requires full deserialization on the server and then reserialization to send the data back to the client.

You can further speed serialization by registering the instantiator for your `DataSerializable` class through `Instantiator`, eliminating the need for reflection to find the right serializer. You can provide your own serialization through the API.

The recommended way to register your custom `Serializer Instantiator` is by specifying it in the serialization-registration element of `cache.xml`.

In addition to speeding standard object serialization, you can use the `DataSerializable` interface to serialize any custom objects you store in the cache.

For more information, see the online Java documentation for `DataSerializable` and `Serializer`.

### Example `cache.xml`:

The following provides an example of how to register an instantiator using `cache.xml`.

```
<serialization-registration>
<instantiator id="30">
    <class-name>com.package.MyClass</class-name>
</instantiator>
</serialization-registration>
```

## Serializing Your Domain Object with DataSerializer

You can also use `DataSerializer` to serialize domain objects. It serializes data in the same way as `DataSerializable` but allows you to serialize classes without modifying the domain class code.

See the [DataSerializer JavaDocs](#) for more information.

## Standard Java Serialization

You can use standard Java serialization for data you only distribute between Java applications. If you distribute your data between non-Java clients and Java servers, you need to do additional programming to get the data between the various class formats.

Standard Java types are serializable by definition. For your domain classes, implement `java.io.Serializable`, then make sure to mark your transient and static variables as needed for your objects. For information, see the online documentation for `java.io.Serializable` for your Java version.

Mixing `DataSerializable` with `Serializable` or `PdxSerializable` use on the same data can result in increased memory use and lower throughput than using just `Serializable` on the entire data, especially if the `Serializable` entries are in collections. The bigger the data collection, the lower the throughput as the metadata for the collection entries is not shared when using `DataSerializable`.



# Chapter 33

---

## Events and Event Handling

vFabric GemFire provides versatile and reliable event distribution and handling for your cached data and system member events.

### How Events Work

Members in your vFabric GemFire distributed system receive cache updates from other members through cache events. The other members can be peers to the member, clients or servers, or other distributed systems.

### Events Features

These are the primary features of GemFire events:

- Content-based events
- Asynchronous event notifications with conflation
- Synchronous event notifications for low latency
- High availability through redundant messaging queues
- Event ordering and once and only-once delivery
- Distributed event notifications
- Durable subscriptions
- Continuous querying

### Types of Events

There are two categories of events and event handlers.

- Cache events in the caching API are used by applications with a cache. Cache events provide detail-level notification for changes to your data. Continuous query events are in this category.
- Administrative events in the administration API are used by administrative applications without caches.

Both kinds of events can be generated by a single member operation.



**Note:** You can handle one of these categories of events in a single system member. You cannot handle both cache and administrative events in a single member.

Because GemFire maintains the order of administrative events and the order of cache events separately, using cache events and administrative events in a single process can cause unexpected results.

### Event Cycle

The following steps describe the event cycle:

1. An operation begins, such as data put or a cache close.

2. The operation execution generates these objects:
  - An object of type `Operation` that describes the method that triggered the event.
  - An event object that describes the event, such as the member and region where the operation originated.
3. The event handlers that can handle the event are called and passed the event objects. Different event types require different handler types in different locations. If there is no matching event handler, that does not change the effect of the operation, which happens as usual.
4. When the handler receives the event, it triggers the handler's callback method for this event. The callback method can hand off the event object as input to another method. Depending on the type of event handler, the callbacks can be triggered before or after the operation. The timing depends on the event handler, not on the event itself.



**Note:** For transactions, after-operation listeners receive the events after the transaction has committed.

5. If the operation is distributed, so that it causes follow-on operations in other members, those operations generate their own events, which can be handled by their listeners in the same way.

## Event Objects

Event objects come in several types, depending on the operation. Some operations generate multiple objects of different types. All event objects contain data describing the event, and each event type carries slightly different kinds of data appropriate to its matching operation. An event object is stable. For example, its content does not change if you pass it off to a method on another thread.

For cache events, the event object describes the operation performed in the local cache. If the event originated remotely, it describes the local application of the remote entry operation, not the remote operation itself. The only exception is when the local region has an empty data policy; then the event carries the information for the remote (originating) cache operation.

## Event Distribution

After a member processes an event in its local cache, it distributes it to remote caches according to the member's configuration and the configurations of the remote caches. For example, if a client updates its cache, the update is forwarded to the client's server. The server distributes the update to its peers and forwards it to any other clients according to their interest in the data entry. If the server system is part of a multi-site deployment and the data region is configured to use a gateway sender, then the gateway sender also forwards the update to a remote site, where the update is further distributed and propagated.

## Event Handlers and Region Data Storage

You can configure a region for no local data storage and still send and receive events for the region. Conversely, if you store data in the region, the cache is updated with data from the event regardless of whether you have any event handlers installed.

## Multiple Listeners

When multiple listeners are installed, as can be done with cache listeners, the listeners are invoked sequentially in the order they were added to the region or cache. Listeners are executed one at a time. So, unless you program a listener to pass off processing to another thread, you can use one listener's work in later listeners.

## Event Ordering

During a cache operation, event handlers are called at various stages of the operation. Some event handlers are called before a region update and some are called after the region update operation. Depending on the type of

event handler being called, the event handler can receive the events in-order or out-of-order in which they are applied on Region.

- `CacheWriter` and `AsyncEventListener` always receive events in the order in which they are applied on region.
- `CacheListener` and `CqListener` can receive events in a different order than the order in which they were applied on the region.



**Note:** An `EntryEvent` contains both the old value and the new value of the entry, which helps to indicate the value that was replaced by the cache operation on a particular key.

## Peer-to-Peer Event Distribution

When a region or entry operation is performed, vFabric GemFire distributes the associated events in the distributed system according to system and cache configurations.

Install a cache listener for a region in each system member that needs to receive notification of region and entry changes.

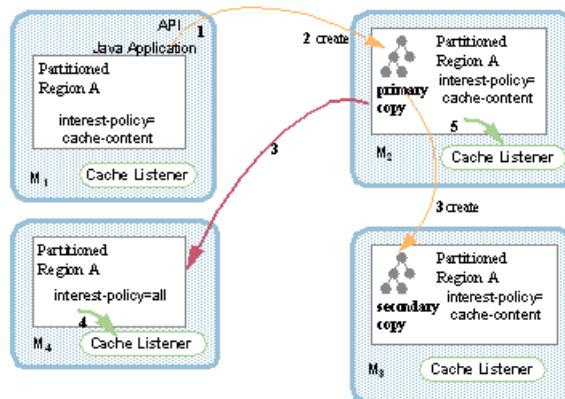
## Events in a Partitioned Region

A distributed operation follows this sequence in a partitioned region:

1. Apply the operation to the cache with the primary data entry, if appropriate.
2. Do the distribution based on the subscription-attributes interest-policy of the other members.
3. Invoke any listeners in the caches that receive the distribution.
4. Invoke the listener in the cache with the primary data entry.

In the following figure:

1. An API call in member M1 creates an entry.
2. The partitioned region creates the new entry in the cache in M2. M2, the holder of the primary copy, drives the rest of the procedure.
3. These two operations occur simultaneously:
  - The partitioned region creates a secondary copy of the entry in the cache in M3. Creating the secondary copy does not invoke the listener on M3.
  - M2 distributes the event to M4. This distribution to the other members is based on their interest policies. M4 has an interest-policy of all, so it receives notification of all events anywhere in the region. Since M1 and M3 have an interest-policy of cache-content, and this event does not affect any pre-existing entry in their local caches, they do not receive the event.
4. The cache listener on M4 handles the notification of the remote event on M2.
5. Once everything on the other members has completed successfully, the original create operation on M2 succeeds and invokes the cache listener on M2.



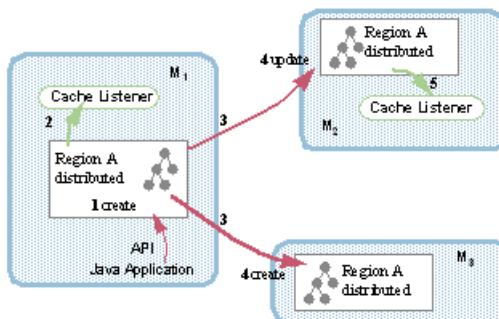
### Events in a Distributed Region

A distributed operation follows this sequence in a distributed region:

1. Apply the operation to the local cache, if appropriate.
2. Invoke the local listeners.
3. Do the distribution.
4. Each member that receives the distribution carries out its own operation in response, which invokes any local listeners.

In the following figure:

1. An entry is created through a direct API call on member M1.
2. The create invokes the cache listener on M1.
3. M1 distributes the event to the other members.
4. M2 and M3 apply the remote change through their own local operations.
5. M3 does a create, but M2 does an update, because the entry already existed in its cache.
6. The cache listener on M2 receives callbacks for the local update. Since there is no cache listener on M3, the callbacks from the create on M3 are not handled. An API call in member M1 creates an entry.



### Managing Events in Multi-threaded Applications

For partitioned regions, GemFire guarantees ordering of events across threads, but for distributed regions it doesn't. For multi-threaded applications that create distributed regions, you need to use your application synchronization to make sure that one operation completes before the next one begins. Distribution through the

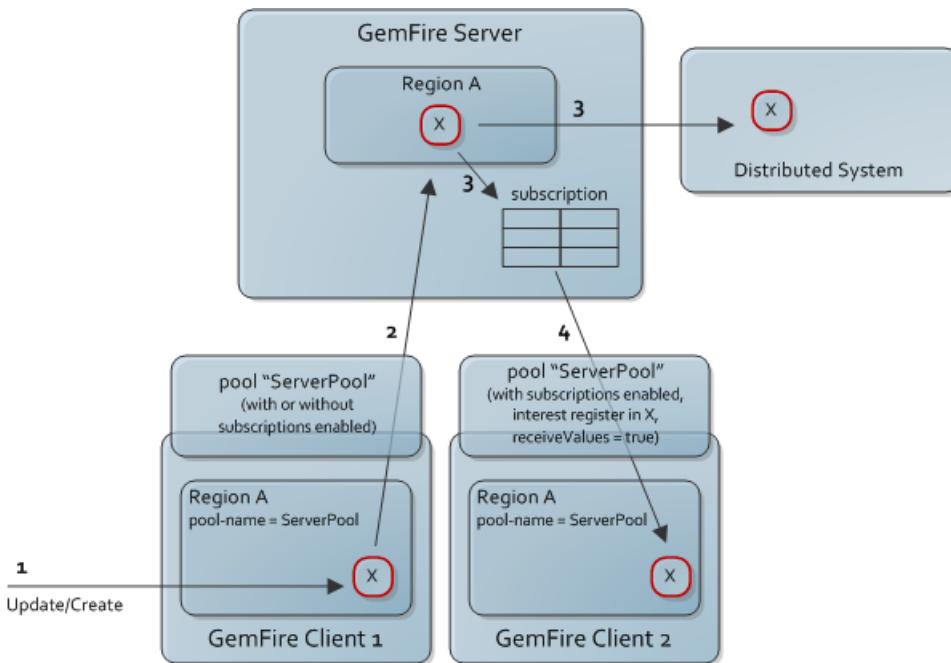
distributed-no-ack queue can work with multiple threads if you set the `conserve-sockets` attribute to true. Then the threads share one queue, preserving the order of the events in distributed regions. Different threads can invoke the same listener, so if you allow different threads to send events, it can result in concurrent invocations of the listener. This is an issue only if the threads have some shared state - if they are incrementing a serial number, for example, or adding their events to a log queue. Then you need to make your code thread safe.

### Client-to-Server Event Distribution

Clients and servers distribute events according to client activities and according to interest registered by the client in server-side cache changes.

When the client updates its cache, changes to client regions are automatically forwarded to the server side. The server-side update is then propagated to the other clients that are connected and have subscriptions enabled. The server does not return the update to the sending client.

The update is passed to the server and then passed, with the value, to every other client that has registered interest in the entry key. This figure shows how a client's entry updates are propagated.



The figure shows the following process:

1. Entry X is updated or created in Region A through a direct API call on Client1.
2. The update to the region is passed to the pool named in the region.
3. The pool propagates the event to the cache server, where the region is updated.
4. The server member distributes the event to its peers and also places it into the subscription queue for Client2 because that client has previously registered interest in entry X.
5. The event for entry X is sent out of the queue to Client2. When this happens is indeterminate.

Client to server distribution uses the client pool connections to send updates to the server. Any region with a named pool automatically forwards updates to the server. Client cache modifications pass first through a client CacheWriter, if one is defined, then to the server through the client pool, and then finally to the client cache itself. A cache writer, either on the client or server side, may abort the operation.

Change in Client Cache	Effect on Server Cache
Entry create or update	Creation or update of entry.
Distributed entry destroy	Entry destroy. The destroy call is propagated to the server even if the entry is not in the client cache.
Distributed region destroy/clear (distributed only)	Region destroy/clear

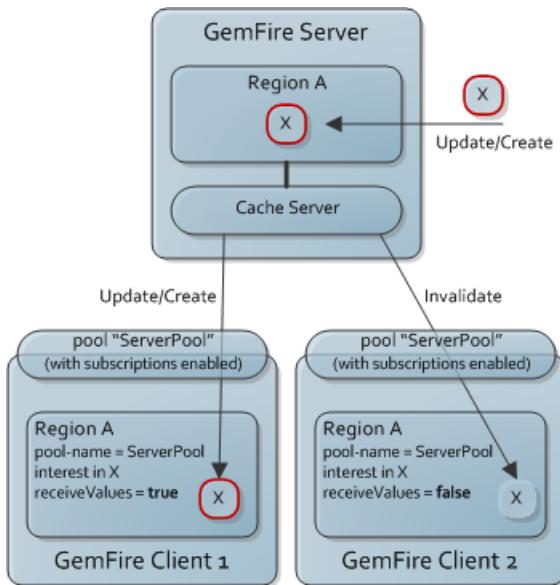


**Note:** Invalidations on the client side are not forwarded to the server.

### Server-to-Client Event Distribution

The server automatically sends entry modification events only for keys in which the client has registered interest. In the interest registration, the client indicates whether to send new values or just invalidations for the server-side entry creates and updates. If invalidation is used, the client then updates the values lazily as needed.

This figure shows the complete event subscription event distribution for interest registrations, with value receipt requested (`receiveValues=true`) and without.



Change in Server Cache	Effect on Client Cache
Entry create/update	For subscriptions with <code>receiveValues</code> set to true, entry create or update. For subscriptions with <code>receiveValues</code> set to false, entry invalidate if the entry already exists in the client cache; otherwise, no effect. The next client get for the entry is forwarded to the server.
Entry invalidate/destroy (distributed only)	Entry invalidate/destroy
Region destroy/clear (distributed only)	Region destroy or local region clear

Server-side distributed operations are all operations that originate as a distributed operation in the server or one of its peers. Region invalidation in the server is not forwarded to the client.

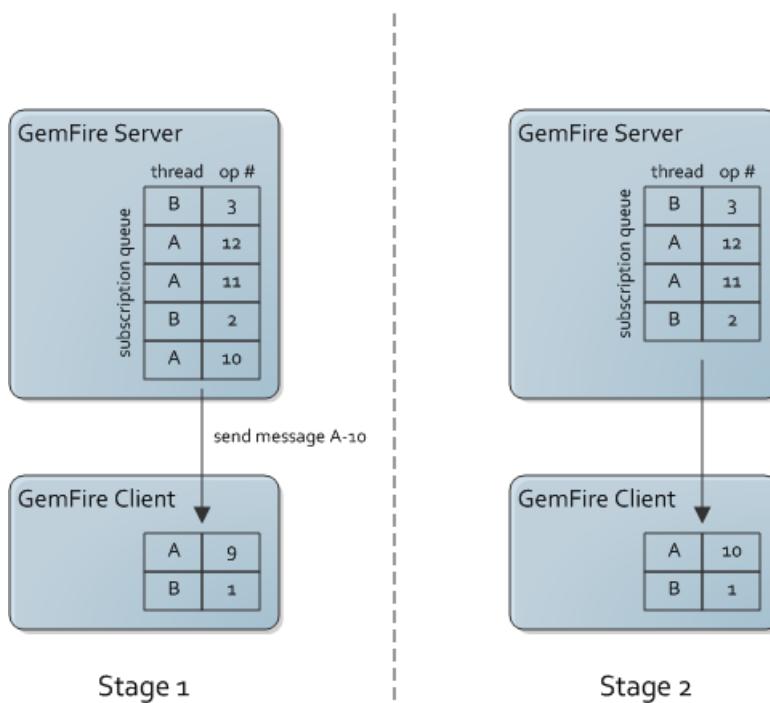


**Note:** To maintain a unified set of data in your servers, do not do local entry invalidation in your server regions.

### Server-to-Client Message Tracking

The server uses an asynchronous messaging queue to send events to its clients. Every event in the queue originates in an operation performed by a thread in a client, a server, or an application in the server's or some other distributed system. The event message has a unique identifier composed of the originating thread's ID combined with its member's distributed system member ID, and the sequential ID of the operation. So the event messages originating in any single thread can be grouped and ordered by time from lowest sequence ID to highest. Servers and clients track the highest sequential ID for each member thread ID.

A single client thread receives and processes messages from the server, tracking received messages to make sure it does not process duplicate sends. It does this using the process IDs from originating threads.



The client's message tracking list holds the highest sequence ID of any message received for each originating thread. The list can become quite large in systems where there are many different threads coming and going and doing work on the cache. After a thread dies, its tracking entry is not needed. To avoid maintaining tracking information for threads that have died, the client expires entries that have had no activity for more than the `subscription-message-tracking-timeout`.

### Client Interest Registration on the Server

The system processes client interest registration following these steps:

1. The entries in the client region that may be affected by this registration are silently destroyed. Other keys are left alone.
  - For the `registerInterest` method, the system destroys all of the specified keys, leaving other keys in the client region alone. So if you have a client region with keys A, B, and C and you register interest in

the key list A, B, at the start of the `registerInterest` operation, the system destroys keys A and B in the client cache but does not touch key C.

- For the `registerInterestRegex` method, the system silently destroys all keys in the client region.
2. The interest specification is sent to the server, where it is added to the client's interest list. The list can specify entries that are not in the server region at the time interest is registered.
  3. If a bulk load is requested in the call's `InterestResultPolicy` parameter, before control is returned to the calling method, the server sends all data that currently satisfies the interest specification. The client's region is updated automatically with the downloaded data. If the server region is partitioned, the entire partitioned region is used in the bulk load. Otherwise, only the server's local cache region is used. The interest results policy options are:
    - **KEYS**—The client receives a bulk load of all available keys matching the interest registration criteria.
    - **KEYS\_VALUES**—The client receives a bulk load of all available keys and values matching the interest registration criteria. This is the default interest result policy.
    - **NONE**—The client does not receive any immediate bulk loading.

Once interest is registered, the server continually monitors region activities and sends events to its clients that match the interest.

- No events are generated by the register interest calls, even if they load values into the client cache..
- The server maintains the union of all of the interest registrations, so if a client registers interest in key 'A', then registers interest in regular expression "B\*", the server will send updates for all entries with key 'A' or key beginning with the letter 'B'.
- The server maintains the interest registration list separate from the region. The list can contain specifications for entries that are not currently in the server region.
- The `registerInterestRegex` method uses the standard `java.util.regex` methods to parse the key specification.

## Server Failover

When a server hosting a subscription queue fails, the queueing responsibilities pass to another server. How this happens depends on whether the new server is a secondary server. In any case, all failover activities are carried out automatically by the GemFire system.

- **Non-HA failover:** The client fails over without high availability if it is not configured for redundancy or if all secondaries also fail before new secondaries can be initialized. As soon as it can attach to a server, the client goes through an automatic reinitialization process. In this process, the failover code on the client side silently destroys all entries of interest to the client and refetches them from the new server, essentially reinitializing the client cache from the new server's cache. For the notify all configuration, this clears and reloads all of the entries for the client regions that are connected to the server. For notify by subscription, it clears and reloads only the entries in the region interest lists. To reduce failover noise, the events caused by the local entry destruction and refetching are blocked by the failover code and do not reach the client cache listeners. Because of this, your clients could receive some out-of-sequence events during and after a server failover. For example, entries that exist on the failed server and not on its replacement are destroyed and never recreated during a failover. Because the destruction events are blocked, the client ends up with entries removed from its cache with no associated destroy events.
- **HA failover:** If your client pool is configured with redundancy and a secondary server is available at the time the primary fails, the failover is invisible to the client. The secondary server resumes queueing activities as soon as the primary loss is detected. The secondary might resend a few events, which are discarded automatically by the client message tracking activities.



**Note:** There is a very small potential for message loss during HA server failover. The risk is not present for failover to secondaries that have fully initialized their subscription queue data. The risk is extremely low in healthy systems that use at least two secondary servers. The risk is higher in unstable systems

where servers often fail and where secondaries do not have time to initialize their subscription queue data before becoming primaries. To minimize the risk, the failover logic chooses the longest-lived secondary as the new primary.



**Note:** Redundancy management is handled by the client, so when a durable client is disconnected from the server, client event redundancy is not maintained. Even if the servers fail one at a time, so that running clients have time to fail over and pick new secondary servers, an offline durable client cannot fail over. As a result, the client loses its queued messages.

## Multi-Site (WAN) Event Distribution

vFabric GemFire distributes a subset of cache events between distributed systems, with a minimum impact on each system's performance. Events are distributed only for regions that you configure to use a gateway sender for distribution.

### Queuing Events for Distribution

In regions that are configured with one or more gateway senders (`gateway-sender-ids` attribute), events are automatically added to a gateway sender queue for distribution to other sites. Events that are placed in a gateway sender queue are distributed asynchronously to remote sites. The ordering of events sent between sites can be preserved using the `order-policy` attribute.

If a queue becomes too full, it is overflowed to disk to keep the member from running out of memory. You can optionally configure the queue to be persisted to disk (with the `is-persistent gateway-sender` attribute). With persistence, if the member that manages the queue goes down, the member picks up where it left off after it restarts.

### Operation Distribution from a Gateway Sender

The multi-site installation is designed for minimal impact on distributed system performance, so only the farthest-reaching entry operations are distributed between sites.

These operations are distributed:

- entry create
- entry put
- entry distributed destroy, providing the operation is not an expiration action

These operations are not distributed:

- get
- invalidate
- local destroy
- expiration actions of any kind
- region operations

### How a Gateway Sender Processes Its Queue

Each primary gateway sender contains a processor thread that reads messages from the queue, batches them, and distributes the batches to a gateway receiver in a remote site. To process the queue, a gateway sender thread takes the following actions:

1. Reads messages from the queue
2. Creates a batch of the messages
3. Synchronously distributes the batch to the other site and waits for a reply

4. Removes the batch from the queue after the other site has successfully replied

Because the batch is not removed from the queue until after the other site has replied, the message cannot get lost. On the other hand, in this mode a message could be processed more than once. If a site goes offline in the middle of processing a batch of messages, then that same batch will be sent again once the site is back online.

You can configure the batch size for messages as well as the batch time interval settings. A gateway sender processes a batch of messages from the queue when either the batch size or the time interval is reached. In an active network, it is likely that the batch size will be reached before the time interval. In an idle network, the time interval will most likely be reached before the batch size. This may result in some network latency that corresponds to the time interval.

### How a Gateway Sender Handles Batch Processing Failure

Exceptions can occur at different points during batch processing:

- The gateway receiver could fail with acknowledgment. If processing fails while the gateway receiver is processing a batch, the receiver replies with a failure acknowledgment that contains the exception, including the identity of the message that failed, and the ID of the last message that it successfully processed. The gateway sender then removes the successfully processed messages and the failed message from the queue and logs an exception with the failed message information. The sender then continues processing the messages remaining in the queue.
- The gateway receiver can fail without acknowledgment. If the gateway receiver does not acknowledge a sent batch, the gateway sender does not know which messages were successfully processed. In this case the gateway sender re-sends the entire batch.
- No gateway receivers may be available for processing. If a batch processing exception occurs because there are no remote gateway receivers available, then the batch remains in the queue. The gateway sender waits for a time, and then attempts to re-send the batch. The time period between attempts is five seconds. The existing server monitor continuously attempts to connect to the gateway receiver, so that a connection can be made and queue processing can continue. Messages build up in the queue and possibly overflow to disk while waiting for the connection.

### List of Event Handlers and Events

vFabric GemFire provides many types of events and event handlers to help you manage your different data and application needs.

#### Event Handlers

Use either cache handlers or admin handlers in any single application. Do not use both. The event handlers in this table are cache handlers unless otherwise noted.

Handler API	Events received	Description
AsyncEventListener	AsyncEvent	Tracks changes in a region for write-behind processing. Extends the CacheCallback interface. You install a write-back cache listener to an AsyncEventQueue instance. You can then add the AsyncEventQueue instance to one or more regions for write-behind processing. See <a href="#">Implementing an AsyncEventListener for Write-Behind Cache Event Handling</a> on page 290.

Handler API	Events received	Description
CacheCallback		Superinterface of all cache event listeners except <code>BridgeMembershipListener</code> . Functions only to clean up resources that the callback allocated.
CacheListener	RegionEvent, EntryEvent	Tracks changes to region and its data entries. Responds synchronously. Extends <code>CacheCallback</code> interface. Installed in region. Receives only local cache events. Install one in every member where you want the events handled by this listener. In a partitioned region, the cache listener only fires in the primary data store. Listeners on secondaries are not fired.
CacheWriter	RegionEvent, EntryEvent	Receives events for <i>pending</i> changes to the region and its data entries in this member or one of its peers. Has the ability to abort the operations in question. Extends <code>CacheCallback</code> interface. Installed in region. Receives events from anywhere in the distributed region, so you can install one in one member for the entire distributed region. Receives events only in primary data store in partitioned regions, so install one in every data store.
GatewayConflictResolver	TimestampedEntryEvent	Decides whether to apply a potentially conflicting event to a region that is distributed over a WAN configuration. This event handler is called only when the distributed system ID of an update event is different from the ID that last updated the region entry.
RegionMembershipListener	RegionEvent	Provides after-event notification when a region with the same name has been created in another member and when other members hosting the region join or leave the distributed system. Extends <code>CacheCallback</code> and <code>CacheListener</code> . Installed in region as a <code>CacheListener</code> .
RegionRoleListener	RoleEvent, RegionEvent, EntryEvent	Tracks the presence of essential membership roles in the distributed system, and changes to member, region, and data entries. Extends <code>CacheCallback</code> , <code>CacheListener</code> , and <code>RegionMembershipListener</code> . Installed in region as a <code>CacheListener</code> .

Handler API	Events received	Description
TransactionListener	TransactionEvent with embedded list of EntryEvent	<p>Tracks the outcome of transactions and changes to data entries in the transaction.</p>  <b>Note:</b> Multiple transactions on the same cache can cause concurrent invocation of TransactionListener methods, so implement methods that do the appropriate synchronizing of the multiple threads for thread-safe operation.
		Extends CacheCallback interface. Installed in cache using transaction manager. Works with region-level listeners if needed.
TransactionWriter	TransactionEvent with embedded list of EntryEvent	Receives events for <i>pending</i> transaction commits. Has the ability to abort the transaction. Extends CacheCallback interface. Installed in cache using transaction manager. At most one writer is called per transaction. Install a writer in every transactional data host.
BridgeMembershipListener	BridgeMembershipEvent	Tracks changes to client/server connections. Static utility, installed at the process level using BridgeMembership.
CqListener	CqEvent	Receives events from the server cache that satisfy a client-specified query. Extends CacheCallback interface. Installed in the client inside a CqQuery.
SystemMembershipListener (administrative listener)	SystemMembershipEvent	Tracks members as they join and leave the distributed system. Installed in any admin application, not installed in the cache. You can call this listener from anywhere in your program.
SystemMemberCacheListener (administrative listener)	SystemMemberCacheEvent SystemMemberRegionEvent	Tracks caches and regions as they come and go. Installed in any admin application, not installed in the cache.

## Cache Events

The events in this table are cache events unless otherwise noted.

Event	Passed to handler ...	Description
AsyncEvent	AsyncEventListener	Provides information about a single event in the cache for asynchronous, write-behind processing.

Event	Passed to handler ...	Description
CacheEvent		Superinterface to <code>RegionEvent</code> and <code>EntryEvent</code> . This defines common event methods, and contains data needed to diagnose the circumstances of the event, including a description of the operation being performed, information about where the event originated, and any callback argument passed to the method that generated this event.
RegionEvent	<code>CacheListener</code> , <code>CacheWriter</code> , <code>RegionMembershipListener</code> , <code>RegionRoleListener</code>	Extends <code>CacheEvent</code> for region events. Provides information about operations that affect the whole region, such as reinitialization of the region after being destroyed.
EntryEvent	<code>CacheListener</code> , <code>CacheWriter</code> , <code>RegionRoleListener</code> , <code>TransactionListener</code> (inside the <code>TransactionEvent</code> )	<p>Extends <code>CacheEvent</code> for entry events. Contains information about an event affecting a data entry in the cache. The information includes the key, the value before this event, and the value after this event.</p> <p><code>EntryEvent.getNewValue</code> returns the current value of the data entry. <code>EntryEvent.getOldValue</code> returns the value before this event if it is available. For a partitioned region, returns the old value if the local cache holds the primary copy of the entry. <code>EntryEvent</code> provides the GemFire transaction ID if available.</p> <p>You can retrieve serialized values from <code>EntryEvent</code> using the <code>getSerialized*</code> methods. This is useful if you get values from one region's events just to put them into a separate cache region. There is no counterpart <code>put</code> function as the <code>put</code> recognizes that the value is serialized and bypasses the serialization step.</p>
RoleEvent	<code>RegionRoleListener</code>	Extends <code>RegionEvent</code> for region reliability events. Contains information about an event where membership roles are lost or gained, which might affect a region's reliability.
TimestampedEntryEvent	<code>GatewayConflictResolver</code>	Extends <code>EntryEvent</code> to include a timestamp and distributed system ID associated with the event. The conflict resolver can compare the timestamp and ID in the event with the values stored in the entry to decide whether the local system should apply the potentially conflicting event.

Event	Passed to handler ...	Description
TransactionEvent	TransactionListener, TransactionWriter	<p>Describes the work done in a transaction. This event may be for a pending or committed transaction, or for the work abandoned by an explicit rollback or failed commit. The work is represented by an ordered list of <code>EntryEvent</code> instances. The entry events are listed in the order in which the operations were performed in the transaction.</p> <p>As the transaction operations are performed, the entry events are conflated, with only the last event for each entry remaining in the list. So if entry A is modified, then entry B, then entry A, the list will contain the event for entry B followed by the second event for entry A.</p>
BridgeMembershipEvent	BridgeMembershipListener	Provides information about changes in a client or server.
CqEvent	CqListener	Provides information about a change to the results of a continuous query running on a server on behalf of a client. <code>CqEvents</code> are processed on the client.
SystemMembershipEvent (administrative event)	SystemMembershipListener	Provides information about member join and departure.
SystemMemberCacheEvent (administrative event)	SystemMemberCacheListener	Provides information about cache create and destroy.
SystemMemberRegionEvent (administrative event)	SystemMemberCacheListener	Provides information about region create and destroy.

## Implementing GemFire Event Handlers

You can specify event handlers for region and region entry operations and for administrative events.

### Implementing Cache Event Handlers

Depending on your installation and configuration, cache events can come from local operations, peers, servers, and remote sites. Event handlers register their interest in one or more events and are notified when the events occur.

For each type of handler, GemFire provides a convenience class with empty stubs for the interface callback methods.



**Note:** Write-behind cache listeners are created by extending the `AsyncEventListener` interface, and they are configured with an `AsyncEventQueue` that you assign to one or more regions. See [Implementing an AsyncEventListener for Write-Behind Cache Event Handling](#) on page 290.

### Procedure

1. Decide which events your application needs to handle. For each region, decide which events you want to handle. For the cache, decide whether to handle transaction events.

2. For each event, decide which handlers to use. The `*Listener` and `*Adapter` classes in `com.gemstone.gemfire.cache.util` show the options.
3. Program each event handler:
  - a. Extend the handler's adapter class.
  - b. If you want to declare the handler in the `cache.xml`, implement the `com.gemstone.gemfire.cache.Declarable` interface as well.
  - c. Implement the handler's callback methods as needed by your application.



**Note:** Improperly programmed event handlers can block your distributed system. Cache events are synchronous. To modify your cache or perform distributed operations based on events, avoid blocking your system by following the guidelines in [How to Safely Modify the Cache from an Event Handler Callback](#) on page 294.

Example:

```
package myPackage;
import com.gemstone.gemfire.cache.Declarable;
import com.gemstone.gemfire.cache.EntryEvent;
import com.gemstone.gemfire.cache.util.CacheListenerAdapter;
import java.util.Properties;

public class MyCacheListener extends CacheListenerAdapter implements Declarable {
    /** Processes an afterCreate event.
     * @param event The afterCreate EntryEvent received */
    public void afterCreate(EntryEvent event) {
        String eKey = event.getKey();
        String eVal = event.getNewValue();
        ... do work with event info
    }
    ... process other event types
}
```

4. Install the event handlers, either through the API or the `cache.xml`.

XML Region Event Handler Installation:

```
<region name="trades">
  <region-attributes ... >
    <!-- Cache listener -->
    <cache-listener>
      <class-name>myPackage.MyCacheListener</class-name>
      <cache-listener>
    </region-attributes>
</region>
```

Java Region Event Handler Installation:

```
RegionFactory rf = cache.createRegionFactory(RegionShortcut.PARTITION);
rf.addCacheListener(new MyCacheListener());
tradesRegion = rf.create("trades");
```

XML Transaction Writer and Listener Installation:

```
<cache search-timeout="60">
  <cache-transaction-manager>
    <transaction-listener>
      <class-name>com.company.data.MyTransactionListener</class-name>
```

```

<parameter name="URL">
    <string>jdbc:cloudscape:rmi:MyData</string>
</parameter>
</transaction-listener>
<transaction-listener>
    .
    .
    </transaction-listener>
<transaction-writer>
    <class-name>com.company.data.MyTransactionWriter</class-name>
<parameter name="URL">
    <string>jdbc:cloudscape:rmi:MyData</string>
</parameter>
<parameter
    .
    .
    </parameter>
</transaction-writer>
</cache-transaction-manager>
    .
    .
</cache>

```

The event handlers are initialized automatically during region creation when you start the member.

### Installing Multiple Listeners on a Region

XML:

```

<region name="exampleRegion">
    <region-attributes>
        .
        .
        <cache-listener>
            <class-name>myCacheListener1</class-name>
        </cache-listener>
        <cache-listener>
            <class-name>myCacheListener2</class-name>
        </cache-listener>
        <cache-listener>
            <class-name>myCacheListener3</class-name>
        </cache-listener>
    </region-attributes>
</region>

```

API:

```

AttributesFactory fac = new
AttributesFactory(this.currRegion.getAttributes());
fac.setScope(Region.DISTRIBUTED_NO_ACK);
CacheListener listener1 = new myCacheListener1();
CacheListener listener2 = new myCacheListener2();
CacheListener listener3 = new myCacheListener3();
fac.initCacheListeners(new CacheListener[] { listener1, listener2,
listener3 });
Region nr = cache.createRegion(name, fac.create());
regionDefaultAttrMap.put(nr.getFullPath(), fac.create());

```

### Implementing an AsyncEventListener for Write-Behind Cache Event Handling

An `AsyncEventListener` receives callbacks for events that change region data. You can use an `AsyncEventListener` implementation as a write-behind cache event handler to synchronize region updates with a database.

## How an AsyncEventListener Works

An `AsyncEventListener` instance is serviced by its own dedicated thread in which a callback method is invoked. Events that update a region are placed in an internal `AsyncEventQueue`, and the dedicated thread dispatches a batch of events at a time to the listener implementation.

You can configure an `AsyncEventQueue` to be either serial or parallel. A serial queue is deployed to one GemFire member, and it delivers all of a region's events in order to a configured `AsyncEventListener` implementation. A parallel queue is deployed to multiple GemFire members, and each instance of the queue simultaneously delivers region events to a local `AsyncEventListener` implementation.

While a parallel queue provides the best throughput for writing events, it provides less control for ordering those events. With a parallel queue, you cannot preserve event ordering for a region as a whole because multiple GemFire servers queue and deliver the region's events at the same time. However, the ordering of events for a given partition (or for a given queue of a distributed region) can be preserved.

For both serial and parallel queues, you can control the maximum amount of memory that each queue uses, as well as the batch size and frequency for processing batches in the queue. You can also configure queues to persist to disk (instead of simply overflowing to disk) so that write-behind caching can pick up where it left off when a member shuts down and is later restarted.

Optionally, a serial queue can use multiple threads to dispatch queued events. When you configure multiple threads for a serial queue, the logical queue that is hosted on a GemFire member is divided into multiple physical queues, each with a dedicated dispatcher thread. You can then configure whether the threads dispatch queued events by key, by thread, or in the same order in which events were added to the queue.



**Note:** You cannot configure multiple dispatcher threads or the ordering policy for a parallel event queue.

## Guidelines for Using an AsyncEventListener

Review the following guidelines before using an `AsyncEventListener`:

- If you use an `AsyncEventListener` to implement a write-behind cache listener, your code should check for the possibility that an existing database connection may have been closed due to an earlier exception. For example, check for `Connection.isClosed()` in a catch block and re-create the connection as needed before performing further operations.
- Use a serial `AsyncEventQueue` if you need to preserve the order of region events when delivering events to your listener implementation. Use parallel queues when the order of events is not important, and when you require maximum throughput for processing events.
- You must install the `AsyncEventListener` implementation on a GemFire member that hosts the region whose events you want to process.
- If you configure a parallel `AsyncEventQueue`, deploy the queue on each GemFire member that hosts the region.
- You can install a listener on more than one member to provide high availability and guarantee delivery for events, in the event that a member with the active `AsyncEventListener` shuts down. At any given time only one member has an active thread for dispatching events. The threads on other members remain on standby for redundancy.
- Install no more than one standby listener (redundancy of at most one) for performance and memory reasons.
- To preserve pending events through member shutdowns, configure GemFire to persist the internal queue of the `AsyncEventListener` to an available disk store. By default, any pending events that reside in the internal queue of an `AsyncEventListener` are lost if the active listener's member shuts down.
- To ensure high availability and reliable delivery of events, configure the event queue to be both persistent and redundant.

## Implementing an AsyncEventListener

To receive region events for processing, you create a class that implements the `AsyncEventListener` interface. The `processEvents` method in your listener receives a list of queued `AsyncEvent` objects in each batch.

Each `AsyncEvent` object contains information about a region event, such as the name of the region where the event occurred, the type of region operation, and the affected key and value.

The basic framework for implementing a write-behind event handler involves iterating through the batch of events and writing each event to a database. For example:

```
class MyAsyncEventListener implements AsyncEventListener {

    public void processEvents(List<AsyncEvent> events) {

        // Process each AsyncEvent

        for(AsyncEvent event: events) {

            // Write the event to a database

        }
    }
}
```

## Processing AsyncEvents

When processing `AsyncEvents`, you can use the `getDeserializedValue` method to obtain cache values for entries that have been updated or created. Since the `getDeserializedValue` method will return a null value for destroyed entries, you should use the `getKey` method to obtain references to cache objects that have been destroyed. Here's an example of processing `AsyncEvents`:

```
public boolean processEvents(@SuppressWarnings("rawtypes") List<AsyncEvent> list)
{
    logger.log (Level.INFO, String.format("Size of List<GatewayEvent> = %s", list.size()));
    List<JdbcBatch> newEntries = new ArrayList<JdbcBatch>();

    List<JdbcBatch> updatedEntries = new ArrayList<JdbcBatch>();
    List<String> destroyedEntries = new ArrayList<String>();
    int possibleDuplicates = 0;

    for (@SuppressWarnings("rawtypes") AsyncEvent ge: list)
    {

        if (ge.getPossibleDuplicate())
            possibleDuplicates++;

        if ( ge.getOperation().equals(Operation.UPDATE))
        {
            updatedEntries.add((JdbcBatch) ge.getDeserializedValue());
        }
        else if ( ge.getOperation().equals(Operation.CREATE))
        {
            newEntries.add((JdbcBatch) ge.getDeserializedValue());
        }
        else if ( ge.getOperation().equals(Operation.DESTROY))
        {
            destroyedEntries.add(ge.getKey().toString());
        }
    }
}
```

```
}
```

```
}
```

## Configuring an AsyncEventListener

To configure a write-behind cache listener, you first configure an asynchronous queue to dispatch the region events, and then create the queue with your listener implementation. You then assign the queue to a region in order to process that region's events.

### Procedure

- Configure a unique `AsyncEventQueue` with the name of your listener implementation. You can optionally configure the queue for parallel operation, persistence, batch size, and maximum memory size. See [List item](#) on page 862 for more information.

#### cache.xml Configuration

```
<cache>
    <async-event-queue id="sampleQueue" persistent="true"
        disk-store-name="exampleStore" parallel="false">
        <async-event-listener>
            <class-name>MyAsyncEventListener</class-name>
            <parameter name="url">
                <string>jdbc:db2:SAMPLE</string>
            </parameter>
            <parameter name="username">
                <string>gfeadmin</string>
            </parameter>
            <parameter name="password">
                <string>admin1</string>
            </parameter>
        </async-event-listener>
    </async-event-queue>
    ...
</cache>
```

#### Java Configuration

```
Cache cache = new CacheFactory().create();
AsyncEventQueueFactory factory = cache.createAsyncEventQueueFactory();
factory.setPersistent(true);
factory.setDiskStoreName("exampleStore");
factory.setParallel(false);
AsyncEventListener listener = new MyAsyncEventListener();
AsyncEventQueue asyncQueue = factory.create("customerWB", listener);
```

- If you are using a parallel `AsyncEventQueue`, repeat the above configuration in each GemFire member that will host the region. Use the same ID and configuration settings for each queue configuration.
- On each GemFire member that hosts the `AsyncEventQueue`, assign the queue to each region that you want to use with the `AsyncEventListener` implementation.

#### cache.xml Configuration

```
<cache>
<region name="data">
    <region-attributes async-event-queue-ids="sampleQueue">
    </region-attributes>
</region>
...
</cache>
```

## Java Configuration

```
RegionFactory rf1 = cache.createRegionFactory();
rf1.addAsyncEventQueue(asyncQueue);
Region customer = rf1.create("Customer");

// Assign the queue to multiple regions as needed
RegionFactory rf2 = cache.createRegionFactory();
Region order = rf2.create("Order");
```

Using the Java API, you can also add and remove queues to regions that have already been created:

```
AttributesMutator mutator = order.getAttributesMutator();
mutator.addAsyncEventQueueId("customerWB");
```

See the GemFire API documentation for more information.

4. Optionally configure persistence and conflation for the queue using the instructions in [Configuring Event Queues](#) on page 309.
5. For serial queues only, optionally configure multiple dispatcher threads and the ordering policy for the queue using the instructions in [Configuring Dispatcher Threads and Order Policy for Event Distribution](#) on page 311.

The `AsyncEventListener` receives events from every region configured with the associated `AsyncEventQueue`.

## How to Safely Modify the Cache from an Event Handler Callback

Event handlers are synchronous. If you need to change the cache or perform any other distributed operation from event handler callbacks, be careful to avoid activities that might block and affect your overall system performance.

## Operations to Avoid in Event Handlers

Do not perform distributed operations of any kind directly from your event handler. If you have any doubt about whether an operation is distributed, please ask GemFire technical support. GemFire is a highly distributed system and many operations that may seem local invoke distributed operations.

These are common distributed operations that can get you into trouble:

- Calling Region methods, on the event's region or any other region.
- Using the GemFire `DistributedLockService`.
- Modifying region attributes.
- Executing a function through the GemFire `FunctionService`.

To be on the safe side, do not make any calls to the GemFire API directly from your event handler. Make all GemFire API calls from within a separate thread or executor.

## How to Perform Distributed Operations Based on Events

If you need to use the GemFire API from your handlers, make your work asynchronous to the event handler. You can spawn a separate thread or use a solution like the `java.util.concurrent.Executor` interface. The `Executor` interface is available in JDK version 1.5 and above.

This example shows a serial executor where the callback creates a `Runnable` that can be pulled off a queue and run by another object. This preserves the ordering of events.

```
public void afterCreate(EntryEvent event) {
    final Region otherRegion = cache.getRegion("/otherRegion");
    final Object key = event.getKey();
    final Object val = event.getNewValue();

    serialExecutor.execute(new Runnable() {
```

```

public void run() {
    try {
        otherRegion.create(key, val);
    }
    catch (com.gemstone.gemfire.cache.RegionDestroyedException e) {
        ...
    }
    catch (com.gemstone.gemfire.cache.EntryExistsException e) {
        ...
    }
}
);
}

```

For additional information on the Executor, see the [SerialExecutor example](#) on the Sun web site.

## Cache Event Handler Examples

Some examples of cache event handlers.

### Declaring and Loading an Event Handler with Parameters

This declares an event handler for a region in the `cache.xml`. The handler is a cache listener designed to communicate changes to a DB2 database. The declaration includes the listener's parameters, which are the database path, username, and password.

```

<region name="exampleRegion">
  <region-attributes>
    .
    .
    <cache-listener>
      <class-name>JDBCListener</class-name>
      <parameter name="url">
        <string>jdbc:db2:SAMPLE</string>
      </parameter>
      <parameter name="username">
        <string>gfeadmin</string>
      </parameter>
      <parameter name="password">
        <string>admin1</string>
      </parameter>
    </cache-listener>
  </region-attributes>
</region>

```

This code listing shows part of the implementation of the JDBCListener declared in the `cache.xml`. This listener implements the `Declarable` interface. When an entry is created in the cache, this listener's `afterCreate` callback method is triggered to update the database. Here the listener's properties, provided in the `cache.xml`, are passed into the `Declarable.init` method and used to create a database connection.

```

. .
public class JDBCListener
extends CacheListenerAdapter
implements Declarable {
  public void afterCreate(EntryEvent e) {
    .
    .
    // Initialize the database driver and connection using input parameters

    Driver driver = (Driver) Class.forName(DRIVER_NAME).newInstance();
    Connection connection =
      DriverManager.getConnection(_url, _username, _password);
    System.out.println(_connection);
    .
  }
}

```

```

    }
    . .
    public void init(Properties props) {
        this._url = props.getProperty("url");
        this._username = props.getProperty("username");
        this._password = props.getProperty("password");
    }
}

```

## Installing an Event Handler Through the API

This listing defines a cache listener using AttributesFactory.

```

AttributesFactory fac = new AttributesFactory();
fac.addCacheListener(new SimpleCacheListener());
Region newReg = this.cache.createRegion(name, fac.create());

```

You can create a cache writer similarly, using the AttributesFactory method setCacheWriter, like this:

```
fac.setCacheWriter(new SimpleCacheWriter());
```

## Installing Multiple Listeners on a Region

XML:

```

<region name="exampleRegion">
    <region-attributes>
        .
        .
        <cache-listener>
            <class-name>myCacheListener1</class-name>
        </cache-listener>
        <cache-listener>
            <class-name>myCacheListener2</class-name>
        </cache-listener>
        <cache-listener>
            <class-name>myCacheListener3</class-name>
        </cache-listener>
    </region-attributes>
</region>

```

API:

```

AttributesFactory fac = new
AttributesFactory(this.currRegion.getAttributes());
fac.setScope(Scope.DISTRIBUTED_NO_ACK);
CacheListener listener1 = new myCacheListener1();
CacheListener listener2 = new myCacheListener2();
CacheListener listener3 = new myCacheListener3();
fac.initCacheListeners(new CacheListener[] { listener1, listener2,
listener3 });
Region nr = cache.createRegion(name, fac.create());
regionDefaultAttrMap.put(nr.getFullPath(), fac.create());

```

## Installing a Write-Behind Cache Listener

```

//AsyncEventQueue with listener that performs WBCL work
<cache>
    <async-event-queue id="sampleQueue" persistent="true"
disk-store-name="exampleStore" parallel="false">
        <async-event-listener>
            <class-name>MyAsyncListener</class-name>
            <parameter name="url">

```

```

        <string>jdbc:db2:SAMPLE</string>
    </parameter>
    <parameter name="username">
        <string>gfeadmin</string>
    </parameter>
    <parameter name="password">
        <string>admin1</string>
    </parameter>
</async-event-listener>
</async-event-queue>

// Add the AsyncEventQueue to region(s) that use the WBCL
<region name="data">
    <region-attributes async-event-queue-ids="sampleQueue">
    </region-attributes>
</region>
</cache>
```

### Installing a BridgeMembershipListener

```

DistributedSystem ds = DistributedSystem.connect(properties);
MyMembershipListenerImpl myListener = new MyMembershipListenerImpl();
BridgeMembership.registerBridgeMembershipListener(myListener);
Cache cache = CacheFactory.create(ds);
```

## Configuring Peer-to-Peer Event Messaging

You can receive events from distributed system peers for any region that is not a local region. Local regions receive only local cache events.

Peer distribution is done according to the region's configuration.

- Replicated regions always receive all events from peers and require no further configuration. Replicated regions are configured using the REPLICATE region shortcut settings.
- For non-replicated regions, decide whether you want to receive all entry events from the distributed cache or only events for the data you have stored locally. To configure:

- To receive all events, set the subscription-attributes interest-policy to all:

```

<region-attributes>
    <subscription-attributes interest-policy="all"/>
</region-attributes>
```

- To receive events just for the data you have stored locally, set the subscription-attributes interest-policy to cache-content or do not set it (cache-content is the default):

```

<region-attributes>
    <subscription-attributes interest-policy="cache-content"/>
</region-attributes>
```

For partitioned regions, this only affects the receipt of events, as the data is stored according to the region partitioning. Partitioned regions with interest policy of all can create network bottlenecks, so if you can, run listeners in every member that hosts the partitioned region data and use the cache-content interest policy.

## Configuring Client/Server Event Messaging

You can receive events from your servers for server-side cache events and query result changes.

For cache updates, you can configure to receive entry keys and values or just entry keys, with the data retrieved lazily when requested. The queries are run continuously against server cache events, with the server sending the deltas for your query result sets.

Before you begin, set up your client/server installation and configure and program your basic event messaging. Servers receive updates for all entry events in their client's client regions.

To receive entry events in the client from the server:

1. Set the client pool subscription-enabled to true. See [<pool>](#) on page 917.
2. Program the client to register interest in the entries you need.



**Note:** This must be done through the API.

Register interest in all keys, a key list, individual keys, or by comparing key strings to regular expressions. By default, no entries are registered to receive updates. Specify whether the server is to send values with entry update events. Interest registration is only available through the API.

1. Get an instance of the region where you want to register interest.
2. Use the regions's `registerInterest*` methods to specify the entries you want. Examples:

```
// Register interest in a single key and download its entry
// at this time, if it is available in the server cache
Region region1 = . . . ;
region1.registerInterest("key-1");

// Register Interest in a List of Keys but do not do an initial bulk
load
// do not send values for create/update events - just send key with
invalidation
Region region2 = . . . ;
List list = new ArrayList();
list.add("key-1");
list.add("key-2");
list.add("key-3");
list.add("key-4");
region2.registerInterest(list, InterestResultPolicy.NONE, false);

// Register interest in all keys and download all available keys now
Region region3 = . . . ;
region3.registerInterest("ALL_KEYS", InterestResultPolicy.KEYS);

// Register Interest in all keys matching a regular expression
Region region1 = . . . ;
region1.registerInterestRegex("[a-zA-Z]+_[0-9]+");
```

You can call the register interest methods multiple times for a single region. Each interest registration adds to the server's list of registered interest criteria for the client. So if a client registers interest in key 'A', then registers interest in regular expression "B\*", the server will send updates for all entries with key 'A' or key beginning with the letter 'B'.

#### Related Topics

[Interest Registration \(GemFire Example\)](#)

3. For highly available event messaging, configure server redundancy. See [Configuring Highly Available Servers](#) on page 299.
4. To have events enqueued for your clients during client downtime, configure durable messaging. See [Implementing Durable Client/Server Messaging](#) on page 301.

5. Write any continuous queries (CQs) that you want to run to receive continuously streaming updates to client queries. CQ events do not update the client cache. See [Implementing Continuous Querying](#) on page 377.



**Note:** If you have dependencies between CQs and/or interest registrations, so that you want the two types of subscription events to arrive as closely together on the client, use a single server pool for everything. Using different pools can lead to time differences in the delivery of events because the pools might use different servers to process and deliver the event messages.

## Configuring Highly Available Servers

With highly-available servers, one of the backups steps in and takes over messaging with no interruption in service if the client's primary server crashes.

To configure high availability, set the `subscription-redundancy` in the client's pool configuration. This setting indicates the number of secondary servers to use. For example:

```
<!-- Run one secondary server -->
<pool name="red1" subscription-enabled="true" subscription-redundancy="1">

    <locator host="nick" port="41111"/>
    <locator host="nora" port="41111"/>
</pool>

<!-- Use all available servers as secondaries. One is primary, the rest are
secondaries -->
<pool name="redX" subscription-enabled="true" subscription-redundancy="-1">

    <locator host="nick" port="41111"/>
    <locator host="nora" port="41111"/>
</pool>
```

When redundancy is enabled, secondary servers maintain queue backups while the primary server pushes events to the client. If the primary server fails, one of the secondary servers steps in as primary to provide uninterrupted event messaging to the client.

The following table describes the different values for the `subscription-redundancy` setting:

subscription-redundancy	Description
0	No secondary servers are configured, so high availability is disabled.
>0	Sets the precise number of secondary servers to use for backup to the primary.
-1	Every server that is not the primary is to be used as a secondary.

## Highly Available Client/Server Event Messaging

With server redundancy, each pool has a primary server and some number of secondaries. The primaries and secondaries are assigned on a per-pool basis and are generally spread out for load balancing, so a single client with multiple pools may have primary queues in more than one server.

The primary server pushes events to clients and the secondaries maintain queue backups. If the primary server fails, one of the secondaries becomes primary to provide uninterrupted event messaging.

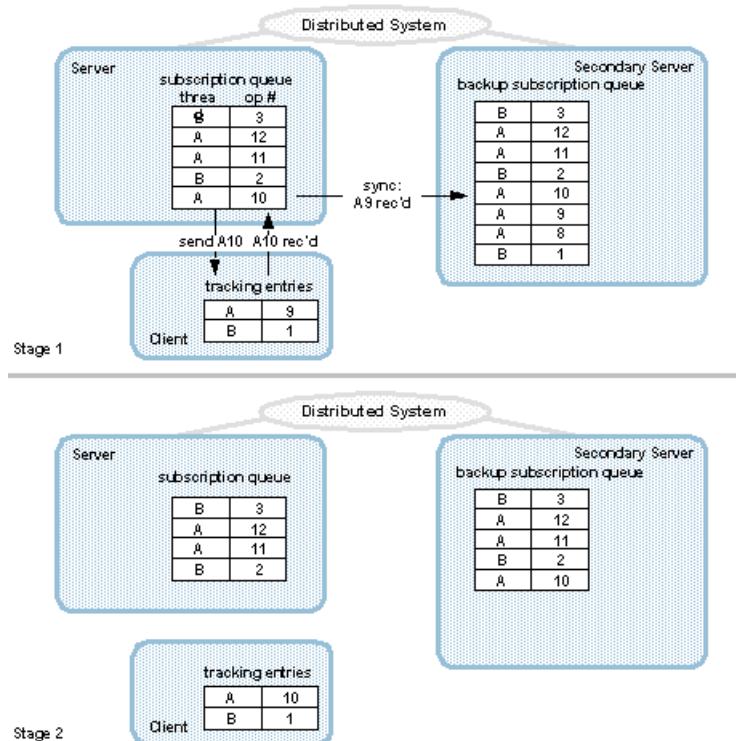
For example, if there are six servers running and `subscription-redundancy` is set to two, one server is the primary, two servers are secondary, and the remaining three do not actively participate in HA for the client. If the primary server fails, the system assigns one of the secondaries as the new primary and attempts to add

another server to the secondary pool to retain the initial redundancy level. If no new secondary server is found, then the redundancy level is not satisfied but the failover procedure completes successfully. As soon as another secondary is available, it is added.

When high availability is enabled:

- The primary server sends event messages to the clients.
- Periodically, the clients send received messages to the server and the server removes the sent messages from its queues.
- Periodically, the primary server synchronizes with its secondaries, notifying them of messages that can be discarded because they have already been sent and received. There is a lag in notification, so the secondary servers remain only roughly synchronized with the primary. Secondary queues contain all messages that are contained in the primary queue plus possibly a few messages that have already been sent to clients.
- In the case of primary server failure, one of the secondaries becomes the primary and begins sending event messages from its queue to the clients. Immediately after failover, the new primary usually resends some messages that were already sent by the old primary. The client recognizes these as duplicates and discards them.

In stage 1 of this figure, the primary sends an event message to the client and a synchronization message to its secondary. By stage 2, the secondary and client have updated their queue and message tracking information. If the primary failed at stage two, the secondary would start sending event messages from its queue beginning with message A10. The client would discard the resend of message A10 and then process subsequent messages as usual.



## Change Server Queue Synchronization Frequency

By default, the primary server sends queue synchronization messages to the secondaries every second. You can change this interval with the server's `cache message-sync-interval`.

Set the interval for queue synchronization messages as follows:

- XML:

```
<!-- Set sync interval to 2 seconds -->
<cache ... message-sync-interval="2" />
```

- Java:

```
cache = CacheFactory.create(system);
cache.setMessageSyncInterval(2);
```

The ideal setting for this interval depends in large part on your application behavior. These are the benefits of shorter and longer interval settings:

- A shorter interval requires less memory in the secondary servers because it reduces queue buildup between synchronizations. In addition, fewer old messages in the secondary queues means reduced message re-sends after a failover. These considerations are most important for systems with high data update rates.
- A longer interval requires fewer distribution messages between the primary and secondary, which benefits overall system performance.

### **Set Frequency of Orphan Removal from the Secondary Queues**

Usually, all event messages are removed from secondary subscription queues based on the primary's synchronization messages. Occasionally, however, some messages are orphaned in the secondary queues. For example, if a primary fails in the middle of sending a synchronization message to its secondaries, some secondaries might receive the message and some might not. If the failover goes to a secondary that did receive the message, the system will have secondary queues holding messages that are no longer in the primary queue. The new primary will never synchronize on these messages, leaving them orphaned in the secondary queues.

To make sure these messages are eventually removed, the secondaries expire all messages that have been enqueued longer than the time indicated by the servers' `message-time-to-live`.

Set the time-to-live as follows:

- XML:

```
<!-- Set message ttl to 5 minutes -->
<cache-server port="41414" message-time-to-live="300" />
```

- Java:

```
Cache cache = ...;
CacheServer cacheServer = cache.addCacheServer();
cacheServer.setPort(41414);
cacheServer.setMessageTimeToLive(200);
cacheServer.start();
```

### **Implementing Durable Client/Server Messaging**

Use durable messaging for subscriptions that you need maintained for your clients even when your clients are down or disconnected. You can configure any of your continuous queries or event subscriptions as durable. Events for durable queries and subscriptions are saved in queue when the client is disconnected and played back when the client reconnects. Other queries and subscriptions are removed from the queue.

Use durable messaging for client/server installations that use continuous queries or event subscriptions.

These are the high-level tasks described in this topic:

1. Configure your client as durable
2. Decide which subscriptions and continuous queries should be durable and configure accordingly

3. Program your client to manage durable messaging for disconnect, reconnect, and event handling

See also [Durable Event Messaging \(GemFire Example\)](#).

### Configure the Client as Durable

Use one of the following methods:

- `gemfire.properties` file:

```
durable-client-id=31
durable-client-timeout=200
```

- Java:

```
Properties props = new Properties();
props.setProperty("durable-client-id", "31");
props.setProperty("durable-client-timeout", " " + 200);
DistributedSystem ds = DistributedSystem.connect(props);
```

The `durable-client-id` indicates that the client is durable and gives the server an identifier to correlate the client to its durable messages. For a non-durable client, this id is an empty string. The ID can be any number that is unique among the clients attached to servers in the same distributed system.

The `durable-client-timeout` tells the server how long to wait for client reconnect. When this timeout is reached, the server stops storing to the client's message queue and discards any stored messages. The default is 300 seconds. This is a tuning parameter. If you change it, take into account the normal activity of your application, the average size of your messages, and the level of risk you can handle, both in lost messages and in the servers' capacity to store enqueued messages. Assuming that no messages are being removed from the queue, how long can the server run before the queue reaches the maximum capacity? How many durable clients can the server handle? To assist with tuning, use the GemFire message queue statistics for durable clients through the disconnect and reconnect cycles.

### Configure Durable Subscriptions and Continuous Queries

The register interest and query creation methods all have an optional boolean parameter for indicating durability. By default all are non-durable.

```
// Durable registration
// Define keySpecification, interestResultPolicy, durability
exampleRegion.registerInterest(keySpecification,
interestResultPolicySpecification, true);

// Durable CQ
// Define cqName, queryString, cqAttributes, durability
CqQuery myCq = queryService.newCq(cqName, queryString, cqAttributes, true);
```

Save only critical messages while the client is disconnected by only indicating durability for critical subscriptions and CQs. When the client is connected to its servers, it receives messages for all keys and queries registered. When the client is disconnected, non-durable interest registrations and CQs are discontinued but all messages already in the queue for them remain there.



**Note:** For a single durable client ID, you must maintain the same durability of your registrations and queries between client runs.

### Program the Client to Manage Durable Messaging

Program your durable client to be durable-messaging aware when it disconnects, reconnects, and handles events from the server.

- Disconnect with a request to keep your queues active by using `Pool.close` or `ClientCache.close` with the boolean `keepalive` parameter.

```
clientCache.close(true);
```



**Note:** To be retained during client down time, durable continuous queries (CQs) must be executing at the time of disconnect.

- Program your durable client's reconnection to:

- Connect, initialize the client cache, regions, any cache listeners, and create and execute any durable continuous queries.
- Run all interest registration calls.



**Note:** Registering interest with `InterestResultPolicy.KEYS_VALUES` initializes the client cache with the *current* values of specified keys. If concurrency checking is enabled for the region, any earlier (older) region events that are replayed to the client are ignored and are not sent to configured listeners. If your client must process all replayed events for a region, register with `InterestResultPolicy.KEYS` or `InterestResultPolicy.NONE` when reconnecting. Or, disable concurrency checking for the region in the client cache. See [Consistency for Region Updates](#) on page 239.

- Call `ClientCache.readyForEvents` so the server will replay stored events. If the ready message is sent earlier, the client may lose events.

```
ClientCache clientCache = ClientCacheFactory.create();
// Here, create regions, listeners, and CQs that are not defined in the
cache.xml . .
// Here, run all register interest calls before doing anything else
clientCache.readyForEvents();
```

- When you program your durable client `CacheListener`:

- Implement the callback methods to behave properly when stored events are replayed. The durable client's `CacheListener` must be able to handle having events played after the fact. Generally listeners receive events very close to when they happen, but the durable client may receive events that occurred minutes before and are not relevant to current cache state.
- Consider whether to use the `CacheListener` callback method, `afterRegionLive`, which is provided specifically for the end of durable event replay. You can use it to perform application-specific operations before resuming normal event handling. If you do not wish to use this callback, and your listener is an instance of `CacheListener` (instead of a `CacheListenerAdapter`) implement `afterRegionLive` as an empty method.

## Durable Client/Server Event Messaging

### Initial Operation

The initial startup of a durable client is similar to the startup of any other client, except that it specifically calls the `ClientCache.readyForEvents` method when all regions, listeners, and CQs on the client are ready to process messages from the server.

### Disconnection

While the client and servers are disconnected, their operation varies depending on the circumstances.

- **Normal disconnect.** When a client closes its connection, the servers stop sending messages to the client and release its connection. If the client requests it, the servers maintain the queues and durable interest list and CQ information until the client reconnects or times out. The non-durable interest lists and CQs are discarded. The servers continue to queue up incoming messages for entries on the durable interest list and in the durable CQ list. All messages that were in the queue when the client disconnected remain in the queue. If the client requests not to have its subscriptions maintained, or if there are no durable subscriptions or CQs, the servers unregister the client and do the same cleanup as for a non-durable client.
- **Abnormal disconnect.** If the client crashes or loses its connections to all servers, the servers automatically maintain its message queue and durable subscriptions and CQs until it reconnects or times out.
- **Client disconnected but operational.** If the client operates while it is disconnected, it gets what data it can from the local client cache. Since updates are not allowed, the data can become stale. An `UnconnectedException` occurs if an update is attempted.
- **Client stays disconnected past timeout period.** The servers track how long to keep a durable subscription queue alive based on the `durable-client-timeout` setting. If the client remains disconnected longer than the timeout, the servers unregister the client and do the same cleanup that is performed for a non-durable client. The servers also log an alert. When a timed-out client reconnects, the servers treat it as a new client making its initial connection.

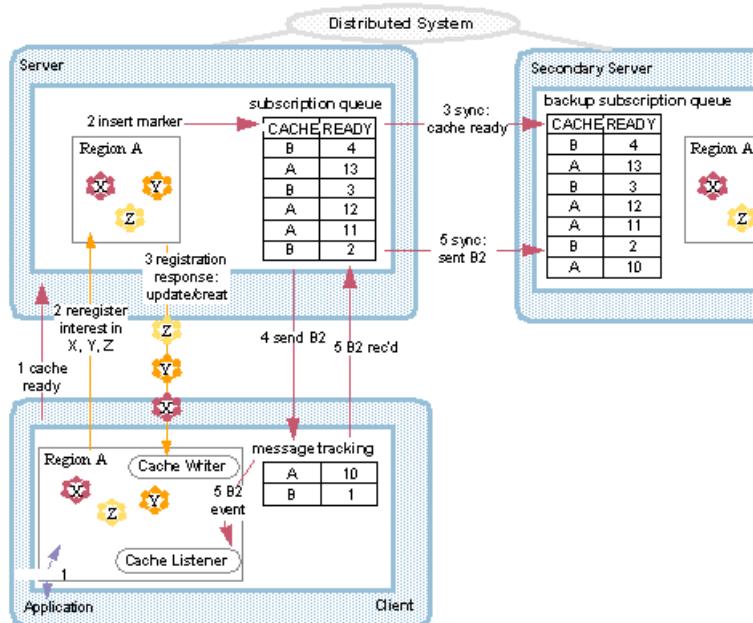
### Reconnection

During initialization, the client cache is not blocked from doing operations, so you might be receiving old stored events from the server at the same time that your client cache is being updated by much more current events. These are the things that can act on the cache concurrently:

- Results returned by the server in response to the client's interest registrations and CQ executions.
- Client cache operations by the application.
- Callbacks triggered by replaying old events from the queue

GemFire handles the conflicts between the application and interest registrations and CQ executions do not create cache update conflicts. But you must program your event handlers so they don't conflict with current operations. This is true for all event handlers, but it is especially important for those used in durable clients. Your handlers may receive events well after the fact and you must ensure your programming takes that into account.

This figure shows the three concurrent procedures during the initialization process. The application begins operations immediately on the client (step 1), while the client's cache ready message (also step 1) triggers a series of queue operations on the servers (starting with step 2 on the primary server). At the same time, the client registers interest (step 2 on the client) and receives a response from the server. Message B2 applies to an entry in Region A, so the cache listener handles B2's event. Because B2 comes before the marker, the client does not apply the update to the cache.



## Durable Event Replay

When a durable client reconnects before the timeout period, the servers replay the events that were stored while the client was gone and then resume normal event messaging to the client. To avoid overwriting current entries with old data, the stored events are not applied to the client cache. Stored events are distinguished from new normal events by a marker that is sent to the client once all old events are replayed.

1. All servers with a queue for this client place a marker in their queue when the client reconnects.
2. The primary server sends the queued messages to the client up to the marker.
3. The client receives the messages but does not apply the usual automatic updates to its cache. If cache listeners are installed, they handle the events.
4. The client receives the marker message indicating that all past events have been played back.
5. The server sends the current list of live regions.
6. For every CacheListener in each live region on the client, the marker event triggers the `afterRegionLive` callback. After the callback, the client begins normal processing of events from the server and applies the updates to its cache.

Even when a new client starts up for the first time, the client cache ready markers are inserted in the queues. If messages start coming into the new queues before the servers insert the marker, those messages are considered as having happened while the client was disconnected, and their events are replayed the same as in the reconnect case.

## Application Operations During Interest Registration

Application operations take precedence over interest registration responses. The client can perform operations while it is receiving its interest registration responses. When adding register interest responses to the client cache, the following rules are applied:

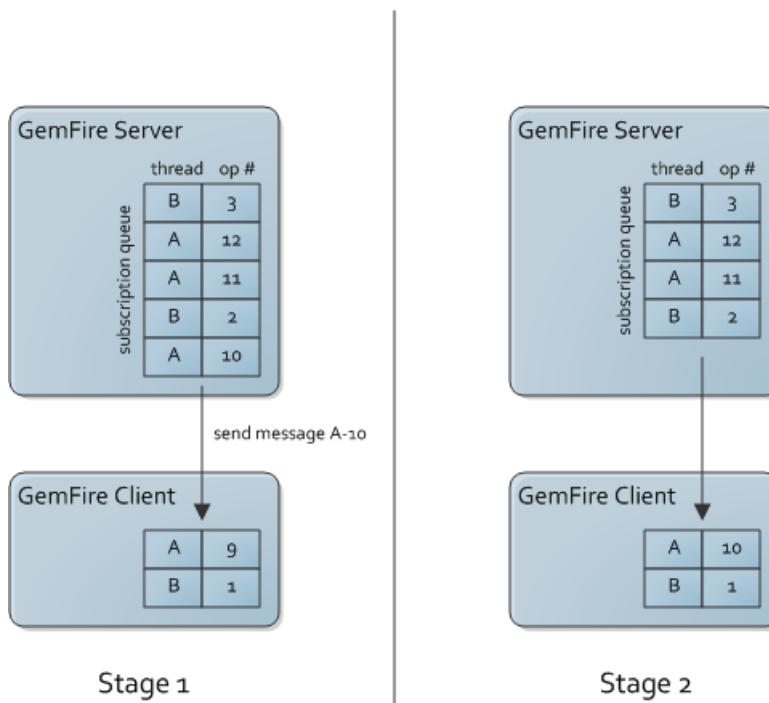
- If the entry already exists in the cache with a valid value, it is not updated.
- If the entry is invalid, and the register interest response is valid, the valid value is put into the cache.
- If an entry is marked destroyed, it is not updated. Destroyed entries are removed from the system after the register interest response is completed.

- If the interest response does not contain any results, because all of those keys are absent from the server's cache, the client's cache can start out empty. If the queue contains old messages related to those keys, the events are still replayed in the client's cache.

### Tuning Client/Server Event Messaging

The server uses an asynchronous messaging queue to send events to its clients. Every event in the queue originates in an operation performed by a thread in a client, a server, or an application in the server's or some other distributed system. The event message has a unique identifier composed of the originating thread's ID combined with its member's distributed system member ID, and the sequential ID of the operation. So the event messages originating in any single thread can be grouped and ordered by time from lowest sequence ID to highest. Servers and clients track the highest sequential ID for each member thread ID.

A single client thread receives and processes messages from the server, tracking received messages to make sure it does not process duplicate sends. It does this using the process IDs from originating threads.



The client's message tracking list holds the highest sequence ID of any message received for each originating thread. The list can become quite large in systems where there are many different threads coming and going and doing work on the cache. After a thread dies, its tracking entry is not needed. To avoid maintaining tracking information for threads that have died, the client expires entries that have had no activity for more than the `subscription-message-tracking-timeout`.

### Conflate the Server Subscription Queue

Conflating the server subscription queue can save space in the server and time in message processing.

Enable conflation at the server level in the server region configuration:

```
<region ... >
  <region-attributes enable-subscription-conflation="true" />
</region>
```

Override the server setting as needed, on a per-client basis, in the client's `gemfire.properties`:

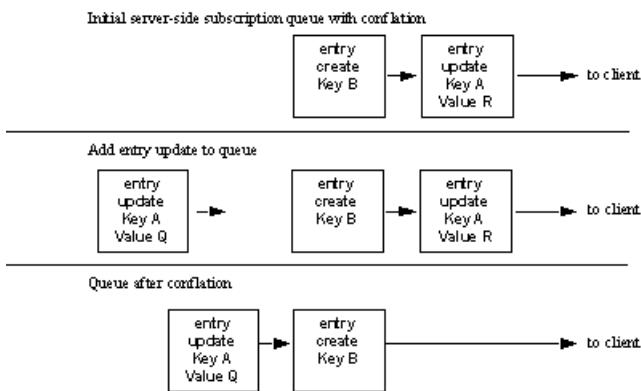
```
conflate-events=false
```

Valid `conflate-events` settings are:

- `server`, which uses the server settings
- `true`, which conflates everything sent to the client
- `false`, which does not conflate anything sent to this client

Conflation can both improve performance and reduce the amount of memory required on the server for queuing. The client receives only the latest available update in the queue for a particular entry key. Conflation is disabled by default.

Conflation is particularly useful when a single entry is updated often and the intermediate updates don't require processing by the client. With conflation, if an entry is updated and there is already an update in the queue for its key, the existing update is removed and the new update is placed at the end of the queue. Conflation is only done on messages that are not in the process of being sent to the client.



**Note:** This method of conflation is different from the one used for multi-site gateway sender queue conflation. It is the same as the method used for the conflation of peer-to-peer distribution messages within a single distributed system.

## Limit the Server's Subscription Queue Memory Use

These are options for limiting the amount of server memory the subscription queues consume.

- Optional: Conflate the subscription queue messages. See [Conflate the Server Subscription Queue](#) on page 306.
- Optional: Increase the frequency of queue synchronization. This only applies to configurations where server redundancy is used for high availability. Increase the client's pool configuration, `subscription-ack-interval`. The client periodically sends a batch acknowledgment of messages to the server, rather than acknowledging each message individually. A lower setting speeds message delivery and generally reduces traffic between the server and client. A higher setting helps contain server queue size. Example:

```
<!-- Set subscription ack interval to 3 seconds -->
<cache>
  <pool ... subscription-enabled="true"
        subscription-ack-interval="3000">
    ...
  </pool>
```

You might want to lower the interval if you have a very busy system and want to reduce the space required in the servers for the subscription queues. More frequent acknowledgments means fewer events held in the server queues awaiting acknowledgment.

- Optional: Limit Queue Size. Cap the server queue size using overflow or blocking. These options help avoid out of memory errors on the server in the case of slow clients. A slow client slows the rate that the server can send messages, causing messages to back up in the queue, possibly leading to out of memory on the server. You can use one or the other of these options, but not both:
  - Optional: Overflow to Disk. Configure subscription queue overflow by setting the server's `client-subscription` properties. With overflow, the most recently used (MRU) events are written out to disk, keeping the oldest events, the ones that are next in line to be sent to the client, available in memory. Example:

```
<!-- Set overflow after 10K messages are enqueued -->
<cache-server port="40404">
  <client-subscription
    eviction-policy="entry"
    capacity="10000"
    disk-store-name="svrOverflow" />
</cache-server>
```

- Optional: Block While Queue Full. Set the server's `maximum-message-count` to the maximum number of event messages allowed in any single subscription queue before incoming messages are blocked. You can only limit the message count, not the size allocated for messages. Examples:

XML:

```
<!-- Set the maximum message count to 50000 entries -->
<cache-server port="41414" maximum-message-count="50000" />
```

API:

```
Cache cache = ...;
CacheServer cacheServer = cache.addCacheServer();
cacheServer.setPort(41414);
cacheServer.setMaximumMessageCount(50000);
cacheServer.start();
```



**Note:** With this setting, one slow client can slow the server and all of its other clients because this blocks the threads that write to the queues. All operations that add messages to the queue block until the queue size drops to an acceptable level. If the regions feeding these queues are partitioned or have distributed-ack or global scope, operations on them remain blocked until their event messages can be added to the queue. If you are using this option and see stalling on your server region operations, your queue capacity might be too low for your application behavior.

## Tune the Client's Subscription Message Tracking Timeout

If the client pool's `subscription-message-tracking-timeout` is set too low, your client will discard tracking records for live threads, increasing the likelihood of processing duplicate events from those threads.

This setting is especially important in systems where it is vital to avoid or greatly minimize duplicate events. If you detect that duplicate messages are being processed by your clients, increasing the timeout may help. Setting `subscription-message-tracking-timeout` may not completely eliminate duplicate entries, but careful configuration can help minimize occurrences.

Duplicates are monitored by keeping track of message sequence IDs from the source thread where the operation originated. For a long-running system, you would not want to track this information for very long periods or the information may be kept long enough for a thread ID to be recycled. If this happens, messages from a new thread

may be discarded mistakenly as duplicates of messages from an old thread with the same ID. In addition, maintaining this tracking information for old threads uses memory that might be freed up for other things.

To minimize duplicates and reduce the size of the message tracking list, set your client `subscription-message-tracking-timeout` higher than double the sum of these times:

- The longest time your originating threads might wait between operations
- For redundant servers add:
  - The server's `message-sync-interval`
  - Total time required for failover (usually 7-10 seconds, including the time to detect failure)

You risk losing live thread tracking records if you set the value lower than this. This could result in your client processing duplicate event messages into its cache for the associated threads. It is worth working to set the `subscription-message-tracking-timeout` as low as you reasonably can.

```
<!-- Set the tracking timeout to 70 seconds -->
<pool name="client" subscription-enabled="true"
subscription-message-tracking-timeout="70000">
  ...
</pool>
```

## Configuring Event Queues

In a multi-site (WAN) installation, GemFire uses gateway sender queues to distribute events for regions that are configured with a gateway sender. AsyncEventListeners also use an asynchronous event queue to distribute events for configured regions. This section describes additional options for configuring the event queues that are used by gateway senders or AsyncEventListener implementations.

Before you begin, set up your multi-site (WAN) installation or configure asynchronous event queues and AsyncEventListener implementations. See [Configuring a Multi-site \(WAN\) System](#) on page 195 or [Implementing an AsyncEventListener for Write-Behind Cache Event Handling](#) on page 290.

### Persisting an Event Queue

You can configure a gateway sender queue or an asynchronous event queue to persist data to disk similar to the way in which replicated regions are persisted.

Persisting a queue provides high availability for the event messaging that the sender performs. For example, if a persistent gateway sender queue exits for any reason, when the member that hosts the sender restarts it automatically reloads the queue and resumes sending messages. If an asynchronous event queue exits for any reason, write-back caching can resume where it left off when the queue is brought back online.

GemFire persists an event queue if you set the `is-persistent` attribute to true. The queue is persisted to the disk store specified in the queue's `disk-store-name` attribute, or to the default disk store if you do not specify a store name.

When you enable persistence for a queue, the `maximum-queue-memory` attribute determines how much memory the queue can consume before it overflows to disk. By default, this value is set to 100MB.



**Note:** If you configure a parallel queue and/or you configure multiple dispatcher threads for a serial queue, the values that are defined in the `maximum-queue-memory` and `disk-store-name` attributes apply to each instance of the queue.

In the example below each gateway sender queue uses "diskStoreA" for persistence and overflow, and the queue has a maximum queue memory of 100MB:

- XML example:

```
<cache>
  <gateway-sender id="persistedsender1" parallel="false"
    remote-distributed-system-id="1"
    is-persistent="true"
    disk-store-name="diskStoreA"
    maximum-queue-memory="100" />
  ...
</cache>
```

- API example:

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

GatewaySenderFactory gateway = cache.createGatewaySenderFactory();
gateway.setParallel(false);
gateway.setPersistenceEnabled(true);
gateway.setDiskStoreName("diskStoreA");
gateway.setMaximumQueueMemory(100);
GatewaySender sender = gateway.create("persistedsender1", "1");
sender.start();
```

If you were to configure 10 dispatcher threads for the above gateway sender, then the total maximum memory for the gateway sender queue would be 1000MB on each GemFire member that hosted the sender.

The following example shows a similar configuration for an asynchronous event queue:

- XML example:

```
<cache>
  <async-event-queue id="persistentAsyncQueue" persistent="true"
    disk-store-name="diskStoreA" parallel="true">
    <async-event-listener>
      <class-name>MyAsyncEventListener</class-name>
      <parameter name="url">
        <string>jdbc:db2:SAMPLE</string>
      </parameter>
      <parameter name="username">
        <string>gfreadmin</string>
      </parameter>
      <parameter name="password">
        <string>admin1</string>
      </parameter>
    </async-event-listener>
  </async-event-queue>
  ...
</cache>
```

- API example:

```
Cache cache = new CacheFactory().create();
AsyncEventQueueFactory factory = cache.createAsyncEventQueueFactory();
factory.setPersistent(true);
factory.setDiskStoreName("diskStoreA");
factory.setParallel(true);
AsyncEventListener listener = new MyAsyncEventListener();
AsyncEventQueue persistentAsyncQueue = factory.create("customerWB",
  listener);
```

## Configuring Dispatcher Threads and Order Policy for Event Distribution

You can use multiple dispatcher threads to process region events simultaneously in a serial gateway sender queue for distribution between sites, or in a serial asynchronous event queue for distributing events for write-behind caching. You can also configure the ordering policy for dispatching those events.



**Note:** Dispatcher threads and ordering policy cannot be configured for parallel gateway sender queues or parallel asynchronous event queues.

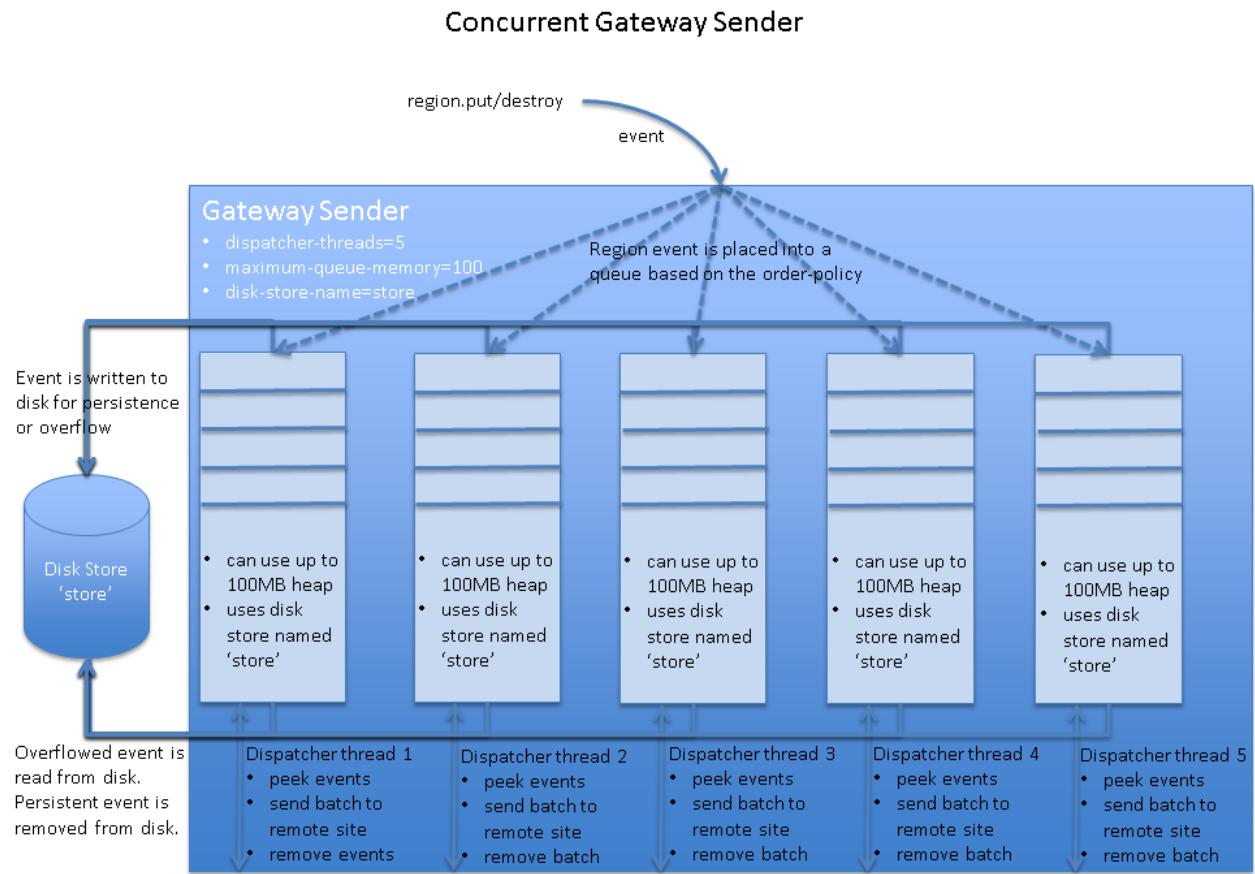
By default, a serial gateway sender queue or asynchronous event queue uses only one dispatcher per queue. However, in some cases an application has the ability to process queued events concurrently for distribution to another GemFire site or listener. In these cases, you can configure multiple dispatcher threads for a single, serial queue.

- [Diagram of Queue with Multiple Dispatcher Threads](#) on page 311
- [Performance and Memory Considerations](#) on page 312
- [Values for Queue Ordering Policy](#) on page 312
- [Examples: Configuring Dispatcher Threads and Ordering Policy for a Serial Gateway Sender Queue](#) on page 313

### Diagram of Queue with Multiple Dispatcher Threads

When you configure multiple dispatcher threads for a serial queue, GemFire creates an additional queue for each thread on each member that hosts the queue. To obtain the maximum throughput, you should increase the number of dispatcher threads until your network is saturated.

The following diagram illustrates a serial gateway sender queue that is configured with multiple dispatcher threads.



## Performance and Memory Considerations

When you configure a serial gateway sender or an asynchronous event queue with multiple dispatcher threads, consider the following:

- Queue attributes are repeated for each queue that is created for a dispatcher thread. That is, each concurrent queue points to the same disk store, so the same disk directories are used. If persistence is enabled and overflow occurs, the threads that insert entries into the queues compete for the disk. This applies to application threads and dispatcher threads, so it can affect application performance.
- The maximum-queue-memory setting applies to each individual queue. If you configure 10 dispatcher threads and the maximum queue memory is set to 100MB, the total maximum queue memory for the queue is 1000MB on each member that hosts the queue.

## Values for Queue Ordering Policy

If you configure multiple dispatcher-threads for a queue, you can also configure the order-policy that those threads use to distribute events from the queue. The valid order policy values are:

- key (default)**. All updates to the same key are distributed in order. GemFire preserves key ordering by placing all updates to the same key in the same dispatcher thread queue. You typically use key ordering when updates to entries have no relationship to each other, such as for an application that uses a single feeder to distribute stock updates to several other systems.

- **thread.** All region updates from a given thread are distributed in order. GemFire preserves thread ordering by placing all region updates from the same thread into the same dispatcher thread queue. In general, use thread ordering when updates to one region entry affect updates to another region entry.
- **partition.** All region events that share the same partitioning key are distributed in order. Specify partition ordering when applications use a [PartitionResolver](#) to implement [Custom-Partition Your Region Data](#) on page 219. With partition ordering, all entries that share the same "partitioning key" (RoutingObject) are placed into the same dispatcher thread queue.

### Examples: Configuring Dispatcher Threads and Ordering Policy for a Serial Gateway Sender Queue

To set the number of dispatcher threads and ordering policy for a serial gateway sender, use one of the following mechanisms.

- **cache.xml configuration**

```
<cache>
  <gateway-sender id="NY" parallel="false"
    remote-distributed-system-id="1"
    is-persistent="true"
    disk-store-name="gateway-disk-store"
    maximum-queue-memory="200"
    dispatcher-threads=5 order-policy="key" />
  ...
</cache>
```

- **Java API configuration**

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

GatewaySenderFactory gateway = cache.createGatewaySenderFactory();
gateway.setParallel(false);
gateway.setPersistenceEnabled(true);
gateway.setDiskStoreName("gateway-disk-store");
gateway.setMaximumQueueMemory(200);
gateway.setDispatcherThreads(5);
gateway.setOrderPolicy(OrderPolicy.KEY);
GatewaySender sender = gateway.create("NY", "1");
sender.start();
```

The following examples show how to set dispatcher threads and ordering policy for an asynchronous event queue:

- **cache.xml configuration**

```
<cache>
  <async-event-queue id="sampleQueue" persistent="true"
    disk-store-name="async-disk-store" parallel="false"
    dispatcher-threads=5 order-policy="key">
    <async-event-listener>
      <class-name>MyAsyncEventListener</class-name>
      <parameter name="url">
        <string>jdbc:db2:SAMPLE</string>
      </parameter>
      <parameter name="username">
        <string>gfeadmin</string>
      </parameter>
      <parameter name="password">
        <string>admin1</string>
      </parameter>
    </async-event-listener>
  </async-event-queue>
</cache>
```

```
</async-event-queue>
...
</cache>
```

- **Java API configuration**

```
Cache cache = new CacheFactory().create();
AsyncEventQueueFactory factory = cache.createAsyncEventQueueFactory();
factory.setPersistent(true);
factory.setDiskStoreName("async-disk-store");
factory.setParallel(false);
factory.setDispatcherThreads(5);
factory.setOrderPolicy(OrderPolicy.KEY);
AsyncEventListener listener = new MyAsyncEventListener();
AsyncEventQueue sampleQueue = factory.create("customerWB", listener);
```

Entry updates in the current, in-process batch are not eligible for conflation.

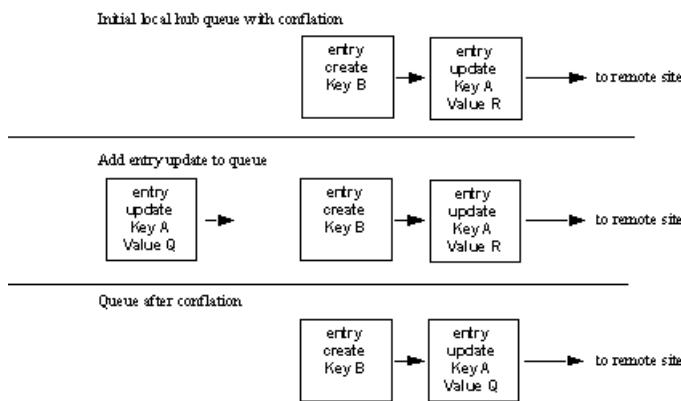
### Conflating Events in a Queue

Conflating a queue improves distribution performance. When conflation is enabled, only the latest queued value is sent for a particular key.



**Note:** Do not use conflation if your receiving applications depend on the specific ordering of entry modifications, or if they need to be notified of every change to an entry.

Conflation is most useful when a single entry is updated frequently, but other sites only need to know the current value of the entry (rather than the value of each update). When an update is added to a queue that has conflation enabled, if there is already an update message in the queue for the entry key, then the existing message assumes the value of the new update and the new update is dropped, as shown here for key A.



**Note:** This method of conflation is different from the one used for server-to-client subscription queue conflation and peer-to-peer distribution within a distributed system.

### Examples: Configuring Conflation for a Gateway Sender Queue

To enable conflation for a gateway sender queue, use one of the following mechanisms:

- **cache.xml configuration**

```
<cache>
  <gateway-sender id="NY" parallel="true">
```

```

remote-distributed-system-id="1"
is-persistent="true"
disk-store-name="gateway-disk-store"
enable-batch-conflation="true"/>
...
</cache>

```

- **Java API configuration**

```

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

GatewaySenderFactory gateway = cache.createGatewaySenderFactory();
gateway.setParallel(true);
gateway.setPersistenceEnabled(true);
gateway.setDiskStoreName("gateway-disk-store");
gateway.setBatchConflationEnabled(true);
GatewaySender sender = gateway.create("NY", "1");
sender.start();

```

Entry updates in the current, in-process batch are not eligible for conflation.

The following examples show how to configure conflation for an asynchronous event queue:

- **cache.xml configuration**

```

<cache>
    <async-event-queue id="sampleQueue" persistent="true"
        disk-store-name="async-disk-store" parallel="false"
        enable-batch-conflation="true">
        <async-event-listener>
            <class-name>MyAsyncEventListener</class-name>
            <parameter name="url">
                <string>jdbc:db2:SAMPLE</string>
            </parameter>
            <parameter name="username">
                <string>gfeadmin</string>
            </parameter>
            <parameter name="password">
                <string>admin1</string>
            </parameter>
        </async-event-listener>
    </async-event-queue>
    ...
</cache>

```

- **Java API configuration**

```

Cache cache = new CacheFactory().create();
AsyncEventQueueFactory factory = cache.createAsyncEventQueueFactory();
factory.setPersistent(true);
factory.setDiskStoreName("async-disk-store");
factory.setParallel(false);
factory.setBatchConflationEnabled(true);
AsyncEventListener listener = new MyAsyncEventListener();
AsyncEventQueue sampleQueue = factory.create("customerWB", listener);

```

Entry updates in the current, in-process batch are not eligible for conflation.



# Chapter 34

## Delta Propagation

Delta propagation allows you to reduce the amount of data you send over the network by including only changes to objects rather than the entire object.

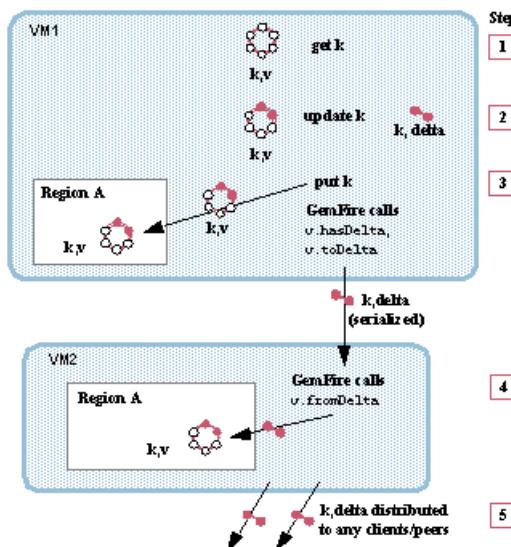
### How Delta Propagation Works

Delta propagation reduces the amount of data you send over the network. You do this by only sending the change, or delta, information about an object, instead of sending the entire changed object. If you do not use cloning when applying the deltas, you can also expect to generate less garbage in your receiving JVMs.

In most distributed data management systems, the data stored in the system tends to be created once and then updated frequently. These updates are sent to other members for event propagation, redundancy management, and cache consistency in general. Tracking only the changes in an updated object and sending only the deltas mean lower network transmission costs and lower object serialization/deserialization costs. Performance improvements can be significant, especially when changes to an object are small relative to its overall size.

vFabric GemFire propagates object deltas using methods that you program. The methods are in the `Delta` interface, which you implement in your cached objects' classes. If any of your classes are plain old Java objects, you need to wrap them for this implementation.

This figure shows delta propagation for a change to an entry with key, k, and value object, v.



1. **get operation.** The get works as usual: the cache returns the full entry object from the local cache or, if it isn't available there, from a remote cache or from a loader.
2. **update methods.** You need to add code to the object's update methods so that they save delta information for object updates, in addition to the work they were already doing.
3. **put operation.** The put works as usual in the local cache, using the full value, then calls hasDelta to see if there are deltas and toDelta to serialize the information. Distribution is the same as for full values, according to member and region configuration. Gateway senders only send full values.
4. **receipt of delta at remote member.** fromDelta extracts the delta information that was serialized by toDelta and applies it to the object in the local cache. The delta is applied directly to the existing value or to a clone, depending on how you configure it for the region.
5. **additional distributions.** As with full distributions, receiving members forward the delta according to their configurations and connections to other members. For example, if VM1 is a client and VM2 is a server, VM2 forwards the delta to its peers and its other clients as needed. Receiving members do not recreate the delta; toDelta is only called in the originating member.

## General Characteristics of Delta Propagation

To use the delta propagation feature, all updates on a key in a region must have value types that implement the `Delta` interface. You cannot mix object types for an entry key where some of the types implement delta and some do not. This is because, when a type implementing the delta interface is received for an update, the existing value for the key is cast to a `Delta` type to apply the received delta. If the existing type does not also implement the `Delta` interface, the operation throws a `ClassCastException`.

Sometimes `fromDelta` cannot be invoked because there is no object to apply the delta to in the receiving cache. When this happens, the system automatically does a full value distribution to the receiver. These are the possible scenarios:

1. If the system can determine beforehand that the receiver does not have a local copy, it sends the initial message with the full value. This is possible when regions are configured with no local data storage, such as with the region shortcut settings `PARTITION_PROXY` and `REPLICATE_PROXY`. These configurations are used to accomplish things like provide data update information to listeners and to pass updates forward to clients.
2. In less obvious cases, such as when an entry has been locally deleted, first the delta is sent, then the receiver requests a full value and that is sent. Whenever the full value is received, any further distributions to the receiver's peers or clients uses the full value.

GemFire also does not propagate deltas for:

- Transactional commit
- The `putAll` operation
- JVMs running GemFire versions that do not support delta propagation (6.0 and earlier)

## Supported Topologies and Limitations

The following topologies support delta propagation (with some limitations):

- **Peer-to-peer.** GemFire system members distribute and receive entry changes using delta propagation, with these requirements and caveats:
  - Regions must be partitioned or have their scope set to `distributed-ack` or `global`. The region shortcut settings for distributed regions use `distributed-ack` scope. Delta propagation does not work for regions with `distributed-no-ack` scope because the receiver could not recover if an exception occurred while applying the delta.
  - For partitioned regions, if a receiving peer does not hold the primary or a secondary copy of the entry, but still requires a value, the system automatically sends the full value.
  - To receive deltas, a region must be non-empty. The system automatically sends the full value to empty regions. Empty regions can send deltas.

- **Client/server.** GemFire clients can always send deltas to the servers, and servers can usually send deltas to clients. These configurations require the servers to send full values to the clients, instead of deltas:
  - When the client's `gemfire.properties` setting `conflate-events` is set to true, the servers send full values for all regions.
  - When the server region attribute `enable-subscription-conflation` is set to true and the client `gemfire.properties` setting `conflate-events` is set to `server`, the servers send full values for the region.
  - When the client region is configured with the PROXY client region shortcut setting (empty client region), servers send full values.
- **Multi-site (WAN).** Gatway senders do not send Deltas. The full value is always sent.

## When to Avoid Delta Propagation

Generally, the larger your objects and the smaller the deltas, the greater the benefits of using delta propagation. Partitioned regions generally benefit more with higher redundancy levels.

By default, delta propagation is enabled in your distributed system.

Delta propagation does not show any significant benefits in some application scenarios. On occasion it results in degradation. These are the main factors that can lower the performance benefits of using delta propagation:

- The added costs of deserializing your objects to apply deltas. Applying a delta requires the entry value to be deserialized. Once this is done, the object is stored back in the cache in serialized form. This aspect of delta propagation only negatively impacts your system if your objects are not already being serialized for other reasons, such as for indexing and querying or for listener operations. Once stored in serialized form, there are reserialization costs for operations that send the object outside of the member, like distribution from a gateway sender, values sent in response to `netSearch` or client requests, and storage to disk. The more operations that require reserialization, the higher the overhead of serializing the object. As with all serialization efforts, you can improve performance in serialization and deserialization by providing custom implementations of `DataSerializable` for your objects.
- Cloning when applying the delta. Using cloning can affect performance and generates extra garbage. Not using cloning is risky however, as you are modifying cached values in place. Without cloning, make sure you synchronize your entry access to keep your cache from becoming inconsistent.
- Problems applying the delta that cause the system to go back to the originator for the full entry value. When this happens, the overall operation costs more than sending the full entry value in the first place. This can be additionally aggravated if your delta is sent to a number of recipients, all or most of them request a full value, and the full value send requires the object to be serialized.
- Disk I/O costs associated with overflow regions. If you use eviction with overflow to disk, on-disk values must be brought into memory in order to apply the delta. This is much more costly than just removing the reference to the disk copy, as you would do with a full value distribution into the cache.

## Delta Propagation Properties

Delta propagation properties can be configured through the API and through the `gemfire.properties` and `cache.xml` files.

### **delta-propagation**

A `gemfire.properties` boolean that enables or disables delta propagation. When false, full entry values are sent for every update. The default setting is true, which enables delta propagation.

Disable delta propagation as follows:

- gemfire.properties:  
delta-propagation=false

- API:

```
Properties props = new Properties();
props.setProperty("delta-propagation", false);
this.cache = new ClientCacheFactory(props).create();
```

### **cloning-enabled**

A region attributes boolean that affects how `fromDelta` applies deltas to the local cache. When true, the updates are applied to a clone of the value and then the clone is saved to the cache. When false, the value is modified in place in the cache. The default value is false.

Exceptions to this behavior:

- If the Cache attribute `copy-on-read` is true, cloning is enabled, regardless of what this attribute is set to.
- Servers running continuous queries (CQs) always clone the existing value and apply the received delta on it, in order to have old and new values for the CQ event. Servers not running CQs use this attribute.

Cloning can be expensive, but it ensures that the new object is fully initialized with the delta before any application code sees it.

When cloning is enabled, by default GemFire does a deep copy of the object, using serialization. You may be able to improve performance by implementing `java.lang.Cloneable` and then implementing the `clone` method, making a deep copy of anything to which a delta may be applied. The goal is to reduce significantly the overhead of copying the object while still retaining the isolation needed for your deltas.

Without cloning:

- It is possible for application code to read the entry value as it is being modified, possibly seeing the value in an intermediate, inconsistent state, with just part of the delta applied. You may choose to resolve this issue by having your application code synchronize on reads and writes.
- GemFire loses any reference to the old value because the old value is transformed in place into the new value. Because of this, your `CacheListener` sees the same new value returned for `EntryEvent.getOldValue` and `EntryEvent.getNewValue`.
- Exceptions thrown from `fromDelta` may leave your cache in an inconsistent state. Without cloning, any interruption of the delta application could leave you with some of the fields in your cached object changed and others unchanged. If you do not use cloning, keep this in mind when you program your error handling in your `fromDelta` implementation.

With cloning:

- The `fromDelta` method generates more garbage in memory.
- Performance is reduced.

Enable cloning as follows:

- `cache.xml`:

```
<region name="region_with_cloning">
    <region-attributes refid="REPLICATE" cloning-enabled="true">
    </region-attributes>
</region>
```

- API:

```
RegionFactory rf = cache.createRegionFactory(REPLICATE);
rf.setCloningEnabled(true);
custRegion = rf.create("customer");
```

## Implementing Delta Propagation

By default, delta propagation is enabled in your distributed system. When enabled, delta propagation is used for objects that implement `com.gemstone.gemfire.Delta`. You program the methods to store and extract delta information for your entries and to apply received delta information.

1. Study your object types and expected application behavior to determine which regions can benefit from using delta propagation. Delta propagation does not improve performance for all data and data modification scenarios. See [When to Avoid Delta Propagation](#) on page 319.
2. For each region where you are using delta propagation, choose whether to enable cloning using the delta propagation property `cloning-enabled`. Cloning is disabled by default. See [Delta Propagation Properties](#) on page 319.
3. If you do not enable cloning, review all associated listener code for dependencies on `EntryEvent.getOldValue`. Without cloning, GemFire modifies the entry in place and so loses its reference to the old value. For delta events, the `EntryEvent` methods `getOldValue` and `getNewValue` both return the new value.
4. For every class where you want delta propagation, implement `com.gemstone.gemfire.Delta` and update your methods to support delta propagation. Exactly how you do this depends on your application and object needs, but these steps describe the basic approach:
  - a. If the class is a plain old Java object (POJO), wrap it for this implementation and update your code to work with the wrapper class.
  - b. Define as transient any extra object fields that you use to manage delta state. This can help performance when the full object is distributed. Whenever standard Java serialization is used, the `transient` keyword indicates to Java to not serialize the field.
  - c. Study the object contents to decide how to handle delta changes. Delta propagation has the same issues of distributed concurrency control as the distribution of full objects, but on a more detailed level. Some parts of your objects may be able to change independent of one another while others may always need to change together. Send deltas large enough to keep your data logically consistent. If, for example, field A and field B depend on each other, then your delta distributions should either update both fields or neither. As with regular updates, the fewer producers you have on a data region, the lower your likelihood of concurrency issues.
  - d. In the application code that puts entries, put the fully populated object into the local cache. Even though you are planning to send only deltas, errors on the receiving end could cause GemFire to request the full object, so you must provide it to the originating put method. Do this even in empty producers, with regions configured for no local data storage. This usually means doing a get on the entry unless you are sure it does not already exist anywhere in the distributed region.
  - e. Change each field's update method to record information about the update. The information must be sufficient for `toDelta` to encode the delta and any additional required delta information when it is invoked.
  - f. Write `hasDelta` to report on whether a delta is available.
  - g. Write `toDelta` to create a byte stream with the changes to the object and any other information `fromDelta` will need to apply the changes. Before returning from `toDelta`, reset your delta state to indicate that there are no delta changes waiting to be sent.
  - h. Write `fromDelta` to decode the byte stream that `toDelta` creates and update the object.
  - i. Make sure you provide adequate synchronization to your object to maintain a consistent object state. If you do not use cloning, you will probably need to synchronize on reads and writes to avoid reading partially written updates from the cache. This synchronization might involve `toDelta`, `fromDelta`, `toData`, `fromData`, and other methods that access or update the object. Additionally, your implementation should take into account the possibility of concurrent invocations of `fromDelta` and one or more of the object's update methods.

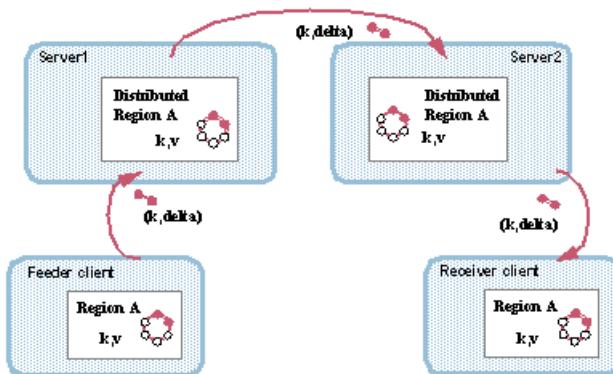
## Errors In Delta Propagation

Errors in delta propagation fall into two categories based on how they are handled by the system:

- Problems applying the delta that can be remedied by requesting the full value in place of the delta. Your put operation does not see errors or exceptions related to this type of delta propagation failure. The system automatically does a full value distribution from the sender to the receiver where the problem occurs. This type of error includes:
  - Unavailable entry value in the receiving cache, either because the entry is missing or its value is null. In both cases, there is nothing to apply the delta to and the full value must be sent. This is most likely to occur if you destroy or invalidate your entries locally, either through application calls or through configured actions like eviction or entry expiration.
  - InvalidDeltaException thrown by `fromDelta` method, programmed by you. This exception enables you to avoid applying deltas that would violate data consistency checks or other application requirements.
  - Any error applying the delta in a client in server-to-client propagation. The client logs a warning in addition to retrieving the full value from the server.
- Problems creating or distributing the delta that cannot be fixed by distributing the full value. In these cases, your put operation fails with an exception. This type of error includes:
  - Error or exception in `hasDelta` or `toDelta`.
  - Error or exception in a server or peer receiver that fall outside of the situations described above in the first category.

## Delta Propagation Example

In this example, the feeder client is connected to the first server, and the receiver client is connected to the second. The servers are peers to each other.



The example demonstrates the following operations:

1. In the Feeder client, the application updates the entry object and puts the entry. In response to the put, GemFire calls `hasDelta`, which returns true, so GemFire calls `toDelta` and forwards the extracted delta to the server. If `hasDelta` returned false, GemFire would distribute the full entry value.
2. In Server1, GemFire applies the delta to the cache, distributes the received delta to the server's peers, and forwards it to any other clients with interest in the entry (there are no other clients to Server1 in this example)
3. In Server2, GemFire applies the delta to the cache and forwards it to its interested clients, which in this case is just the Receiver client.

This example shows the basic approach to programming a Delta implementation.

```

package delta;

import com.gemstone.gemfire.Delta;
import com.gemstone.gemfire.InvalidDeltaException;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.io.Serializable;

/**
 * Sample implementation of Delta
 *
 * @author GemStone Systems, Inc.
 * @since 6.1
 */
public class SimpleDelta implements Delta, Serializable {

    // Original object fields
    private int intValue;
    private double doubleVal;

    // Added for delta - one boolean per field to track changed status
    private transient boolean intFldChd = false;
    private transient boolean dblFldChd = false;

    public SimpleDelta(){}
    public SimpleDelta(int i, double d){
        this.intValue = i;
        this.doubleVal = d;
    }

    public boolean hasDelta() {
        return this.intFldChd || this.dblFldChd;
    }

    public void toDelta(DataOutput out) throws IOException {
        System.out.println("Extracting delta from " + this.toString());
        // Write information on what has changed to the
        // data stream, so fromDelta knows what it's getting
        out.writeBoolean(intFldChd);
        if (intFldChd) {
            // Write just the changes into the data stream
            out.writeInt(this.intValue);
        }
        // Once the delta information is written, reset the delta status field
        this.intFldChd = false;
        System.out.println(" Extracted delta from field 'intValue' = "
                + this.intValue);
    }
    out.writeBoolean(dblFldChd);
    if (dblFldChd) {
        out.writeDouble(this.doubleVal);
        this.dblFldChd = false;
        System.out.println(" Extracted delta from field 'doubleVal' = "
                + this.doubleVal);
    }
}

public void fromDelta(DataInput in) throws IOException,

```

```
InvalidDeltaException {
    System.out.println("Applying delta to " + this.toString());
    // For each field, read whether there is a change
    if (in.readBoolean()) {
        // Read the change and apply it to the object
        this.intVal = in.readInt();
        System.out.println(" Applied delta to field 'intVal' = "
            + this.intVal);
    }
    if (in.readBoolean()) {
        this.doubleVal = in.readDouble();
        System.out.println(" Applied delta to field 'doubleVal' = "
            + this.doubleVal);
    }
}
// In the setter methods, add setting of delta-related
// fields indicating what has changed
public void setIntVal(int anIntVal) {
    this.intFldChd = true;
    this.intVal = anIntVal;
}

public void setDoubleVal(double aDoubleVal) {
    this.dblFldChd = true;
    this.doubleVal = aDoubleVal;
}

public String toString() {
    return "SimpleDelta [ hasDelta = " + hasDelta() + ", intVal = " +
        this.intVal + ", doubleVal = {" + this.doubleVal + "} ]";
}
```

# Chapter 35

---

## Querying

Since GemFire regions are key-value stores where values can range from simple byte arrays to complex nested objects, GemFire uses a query syntax based on OQL (Object Query Language) to query region data. OQL is very similar to SQL, but OQL allows you to query complex objects, object attributes, and methods.

While OQL and SQL have many syntactical similarities, they have significant differences as well. For example, while OQL does not offer all of the capabilities of SQL such as aggregates, it does give you the ability to execute queries on complex object graphs, query object attributes and invoke object methods.

### GemFire Querying FAQ and Examples

This topic answers some frequently asked questions on querying functionality. It provides examples to help you get started with GemFire querying.

This topic answers some frequently asked questions on querying functionality. It provides examples to help you get started with GemFire querying.

For additional information on GemFire querying, see [Querying](#) on page 325.

- [How do I write and execute a query against a GemFire region?](#) on page 325
- [Can I see query string examples, listed by query type?](#) on page 326
- [Which APIs should I use to write my queries?](#) on page 333
- [How do I invoke an object's method in a query?](#) on page 333
- [Can I invoke a static method on an object in a query?](#) on page 333
- [How do I write a reusable query?](#) on page 333
- [When should I create indexes to use in my queries?](#) on page 334
- [How do I create an indexes?](#) on page 334
- [Can I query a partitioned region? Can I perform a join query on a partitioned region?](#) on page 335
- [How can I improve the performance of a partitioned region query?](#) on page 335
- [Which query language elements are supported in GemFire?](#) on page 335
- [How do I debug queries?](#) on page 335

#### How do I write and execute a query against a GemFire region?

To write and execute a query in GemFire, you can use any of the following mechanisms. Sample query code follows.

- GemFire querying APIs ([Java APIs](#) or [Native Client C++ API](#) or [.NET C# API](#))
- [gfshgfh](#) (pronounced "gee - fish" ) is a vFabric GemFire command line tool for browsing and editing data stored in GemFire. Its rich set of Unix-flavored commands allows you to easily access data, monitor peers, redirect outputs to files, and run batch scripts. command-line interface
- graphical-based [Data Browser Tool](#)

**Sample GemFire Query Code (Java)**

```
// Identify your query string.
String queryString = "SELECT * FROM /exampleRegion";

// Get QueryService from Cache.
QueryService queryService = cache.getQueryService();

// Create the Query Object.
Query query = queryService.newQuery(queryString);

// Execute Query locally. Returns results set.
SelectResults results = (SelectResults)query.execute();

// Find the Size of the ResultSet.
int size = results.size();

// Iterate through your ResultSet.
Portfolio p = (Portfolio)results.iterator().next(); /* Region containing
Portfolio object. */
```

**Sample GemFire Native Client Query Code***C# .NET Sample*

```
Query<Portfolio> qry = qrySvc.NewQuery("SELECT * FROM /exampleRegion");
ISelectResults<Portfolio> results = qry.Execute();
SelectResultsIterator<Portfolio> iter = results.GetIterator(); while
(iter.MoveNext()) {
    Console.WriteLine( iter.Current.ToString()); }
```

*C++ Sample*

```
QueryServicePtr qrySvcPtr =
    cachePtr->getQueryService("examplePool");
QueryPtr qry = qrySvcPtr->newQuery(
    "SELECT * FROM /exampleRegion");
SelectResultsPtr resultsPtr = qry->execute(10);
SelectResultsIterator iter = resultsPtr->getIterator();
while(iter.hasNext())
{
    PortfolioPtr portfolio = dynCast<PortfolioPtr > (iter.next());
}
```

**Can I see query string examples, listed by query type?**

The following example query strings use the /exampleRegion whose keys are the portfolio ID and whose values correspond to the summarized data shown in the following class definitions:

```
class Portfolio implements DataSerializable {
    int ID;
    String type;
    String status;
    Map positions;
}
class Position implements DataSerializable {
    String secId;
    double mktValue;
    double qty;
}
```

## vFabric GemFire Query String Examples

### Basic WHERE Clause Examples

In the following examples, the status field is type String and the ID field is type int. See [Supported Literals](#) on page 354 for a complete list of literals supported in GemFire querying.

Select all active portfolios.

```
SELECT * FROM /exampleRegion WHERE status = 'active'
```

Select all portfolios whose status begins with 'activ'.

```
SELECT * FROM /exampleRegion p WHERE p.status LIKE 'activ%'
```

Select all portfolios whose ID is greater than 100.

```
SELECT * from /exampleRegion p WHERE p.ID > 100
```

### Using DISTINCT

Select distinct Objects from the region that satisfy the where clause condition of status = 'active'.

```
SELECT DISTINCT * FROM /exampleRegion WHERE status = 'status'
```

### Aliases and Synonyms.

In the query string, the path expressions (region and its objects) can be defined using an alias. This alias can be used or referred to in other places in the query.

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.status = 'active'
```

```
SELECT p.ID, p.status FROM /exampleRegion p WHERE p.ID > 0
```

### Using the NOT Operator.

See [Operators](#) on page 352 for a complete list of supported operators.

```
SELECT DISTINCT * FROM /exampleRegion WHERE NOT (status = 'active') AND ID = 2
```

```
SELECT * FROM /exampleRegion WHERE NOT (ID IN SET(1,2))
```

### Using the AND and OR Operators.

See [Operators](#) on page 352 for a complete list of supported operators.

```
SELECT * FROM /exampleRegion WHERE ID > 4 AND ID < 9
```

```
SELECT * FROM /exampleRegion WHERE ID = 0 OR ID = 1
```

```
SELECT DISTINCT p.status FROM /exampleRegion p WHERE (p.createTime IN SET (10|) OR p.status IN SET ('active')) AND p.ID > 0
```

**vFabric GemFire Query String Examples****Using not equal to**

```
SELECT * FROM /exampleRegion portfolio WHERE portfolio.ID <> 2
```

```
SELECT * FROM /exampleRegion portfolio WHERE portfolio.ID != 2
```

## Projection attribute example

```
SELECT p.get('account') FROM /exampleRegion p
```

**Querying nested collections.**

The following query uses Positions of type HashMap.

```
SELECT p, pos FROM /exampleRegion p, p.positions.values pos WHERE pos.secId = 'VMW'
```

**Using LIMIT**

```
SELECT * FROM /exampleRegion p WHERE p.ID > 0 LIMIT 2
```

**Using COUNT**

See [COUNT](#) on page 348 for more information.

```
SELECT COUNT(*) FROM /exampleRegion WHERE ID > 0
```

```
SELECT COUNT(*) FROM /exampleRegion WHERE ID > 0 LIMIT 50
```

```
SELECT COUNT(*) FROM /exampleRegion WHERE ID > 0 AND status LIKE 'act%'
```

```
SELECT COUNT(*) FROM /exampleRegion WHERE ID IN SET(1,2,3,4,5)
```

```
SELECT COUNT(*) FROM /exampleRegion p, p.positions.values pos WHERE p.ID > 0 AND pos.secId 'IBM'
```

```
SELECT DISTINCT COUNT(*) FROM /exampleRegion p, p.positions.values pos WHERE p.ID > 0 OR p.status = 'active' OR pos.secId OR pos.secId = 'IBM'
```

**Using LIKE.** Note that you cannot use indexes when using LIKE.

```
SELECT * FROM /exampleRegion ps WHERE ps.pkid LIKE '_bc'
```

```
SELECT * FROM /exampleRegion ps WHERE ps.status LIKE '_b_' OR ps.pkid = '2'
```

```
SELECT * FROM /exampleRegion ps WHERE ps.status LIKE '%b%
```

**vFabric GemFire Query String Examples****Using Region Entry Keys and Values**

```
SELECT * FROM /exampleRegion.keys k WHERE k.ID = 1
```

```
SELECT key, positions FROM /exampleRegion.entrySet, value.positions.values
positions
WHERE positions.mktValue >= 25.00
```

```
SELECT DISTINCT entry.value FROM /exampleRegion.entries entry WHERE
entry.key = '1'
```

```
SELECT * FROM /exampleRegion.entries entry WHERE entry.value.ID > 1
```

```
SELECT entry.value FROM /exampleRegion.entries entry WHERE entry.key = '1'
```

```
SELECT * FROM /exampleRegion.keySet key WHERE key = '1'
```

```
SELECT * FROM /exampleRegion.values portfolio
WHERE portfolio.status = 'active'
```

**Nested Queries**

```
IMPORT "query".Portfolio;
SELECT * FROM /exampleRegion, (SELECT DISTINCT * FROM /exampleRegion p
TYPE Portfolio, p.positions
WHERE value!=null)
```

```
SELECT DISTINCT * FROM (SELECT DISTINCT * FROM /exampleRegion portfolios,
positions pos)
WHERE pos.value.secId = 'IBM'
```

```
SELECT * FROM /exampleRegion portfolio
WHERE portfolio.ID IN (SELECT p2.ID FROM /exampleRegion2 p2 WHERE p2.ID >
1)
```

```
SELECT DISTINCT * FROM /exampleRegion p, (SELECT DISTINCT pos
FROM /exampleRegion x, x.positions.values pos WHERE x.ID = p.ID ) AS itrX
```

**Query the results of a FROM clause expression**

```
SELECT DISTINCT * FROM (SELECT DISTINCT * FROM /Portfolios ptf, positions
pos) p
WHERE p.get('pos').value.secId = 'IBM'
```

**Hash Map Query**

**vFabric GemFire Query String Examples**

Query using a hashmap. In the following examples, 'version' is one of the keys in the hashmap.

```
SELECT * FROM /exampleRegion p WHERE p['version'] = '1.0'
```

```
SELECT entry.key, entry.value FROM /exampleRegion.entries entry
WHERE entry.value['version'] = '100'
```

**Map example where "map" is a nested HashMap object**

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.portfolios['key2'] >= 3
```

**Example Queries that Fetch Array Values**

```
SELECT * FROM /exampleRegion p WHERE p.names[0] = 'aaa'
```

```
SELECT * FROM /exampleRegion p WHERE p.collectionHolderMap.get('1').arr[0]
= '0'
```

**Using ORDER BY (and ORDER BY with LIMIT)**

You must use the DISTINCT keyword with ORDER BY queries.

```
SELECT DISTINCT * FROM /exampleRegion WHERE ID < 101 ORDER BY ID
```

```
SELECT DISTINCT * FROM /exampleRegion WHERE ID < 101 ORDER BY ID asc
```

```
SELECT DISTINCT * FROM /exampleRegion WHERE ID < 101 ORDER BY ID desc
```

```
SELECT DISTINCT key.ID, key.status AS st FROM /exampleRegion.keys key
WHERE key.status = 'inactive' ORDER BY key.status desc, key.ID LIMIT 1
```

```
SELECT DISTINCT * FROM /exampleRegion p ORDER BY p.getP1().secId, p.ID
dec, p.ID LIMIT 9
```

```
SELECT DISTINCT * FROM /exampleRegion p ORDER BY p.ID, val.secId LIMIT 1
```

```
SELECT DISTINCT e.key FROM /exampleRegion.entrySet e ORDER BY e.key.ID
desc, e.key.pkid desc
```

```
SELECT DISTINCT p.names[1] FROM /exampleRegion p ORDER BY p.names[1]
```

**vFabric GemFire Query String Examples****Join Queries**

```
SELECT * FROM /exampleRegion portfolio1, /exampleRegion2 portfolio2
WHERE portfolio1.status = portfolio2.status
```

```
SELECT portfolio1.ID, portfolio2.status FROM /exampleRegion portfolio1,
/exampleRegion2 portfolio2
WHERE portfolio1.status = portfolio2.status
```

```
SELECT * FROM /exampleRegion portfolio1, portfolio1.positions.values
positions1,
/exampleRegion2 portfolio2, portfolio2.positions.values positions2 WHERE
positions1.secId = positions1.secId
```

```
SELECT * FROM /exampleRegion portfolio1, portfolio1.positions.values
positions1,
/exampleRegion2 portfolio2, portfolio2.positions.values positions2 WHERE
portfolio1.ID = 1
AND positions1.secId = positions1.secId
```

```
SELECT DISTINCT a, b.price FROM /exampleRegion1 a, /exampleRegion2 b WHERE
a.price = b.price
```

**Using AS**

```
SELECT * FROM /exampleRegion p, p.positions.values AS pos WHERE pos.secId
!= '1'
```

**Using TRUE**

```
SELECT DISTINCT * FROM /Portfolios WHERE TRUE
```

**Using IN and SET**

See also [IN and SET](#) on page 343.

```
SELECT * FROM /exampleRegion portfolio WHERE portfolio.ID IN SET(1, 2)
```

```
SELECT * FROM /exampleRegion portfolio, portfolio.positions.values positions
WHERE portfolio.Pk IN SET ('1', '2') AND positions.secId = '1'
```

```
SELECT * FROM /exampleRegion portfolio, portfolio.positions.values positions
WHERE portfolio.Pk IN SET ('1', '2') OR positions.secId IN SET ('1', '2',
'3')
```

```
SELECT * FROM /exampleRegion portfolio, portfolio.positions.values positions
WHERE portfolio.Pk IN SET ('1', '2') OR positions.secId IN SET ('1', '2',
```

**vFabric GemFire Query String Examples**

```
'3')
AND portfolio.status = 'active'
```

**Querying for Set values**

In the following query, sp is of type Set.

```
SELECT * FROM /exampleRegion WHERE sp = set('20', '21', '22')
```

If the Set (sp) only contains '20' and '21', then the query will evaluate to false. The query compares the two sets and looks for the presence of elements in both sets.

For other collection types like list (sp is of type List), the query can be written as follows:

```
SELECT * FROM /exampleRegion WHERE sp.containsAll(set('20', '21', '22'))
```

**Invoking Methods on Objects**

See [Method Invocations](#) on page 342 for more information.

```
SELECT * FROM /exampleRegion p WHERE p.length > 1
```

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.positions.size >= 2
```

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.positions.isEmpty
```

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.name.startsWith('Bo')
```

**Using Query-Level Debugging**

To set debugging on the query level, add the <trace> keyword before the query. (If you are using an IMPORT statement, include it before the IMPORT).

```
<trace>
```

```
SELECT * from /exampleRegion, positions.values TYPE myclass
```

**Using Reserved Words in Queries**

To access any method, attribute, or named object that has the same name as a query language reserved word, enclose the name within double quotation marks.

```
SELECT * FROM /exampleRegion WHERE status = 'active' AND "type" = 'XYZ'
```

```
SELECT DISTINCT "type" FROM /exampleRegion WHERE status = 'active'
```

**Using IMPORT**

In the case where the same class name resides in two different namespaces (packages), there needs to be a means of referring to different classes of the same name. The IMPORT statement is used to establish a namespace for a class in a query.

```
IMPORT package.Position;
SELECT DISTINCT * FROM /exampleRegion, positions.values positions TYPE
Position WHERE positions.mktValue >= 25.00
```

## vFabric GemFire Query String Examples

### Using TYPE

Specifying object type helps the query engine to process the query at optimal speed. Apart from specifying the object types during configuration (using key-constraint and value-constraint), type can be explicitly specified in the query string.

```
SELECT DISTINCT * FROM /exampleRegion, positions.values positions TYPE Position WHERE positions.mktValue >= 25.00
```

### Using ELEMENT

Using ELEMENT(expr) extracts a single element from a collection or array. This function throws a FunctionDomainException if the argument is not a collection or array with exactly one element.

```
ELEMENT(SELECT DISTINCT * FROM /exampleRegion WHERE id = 'XYZ-1').status = 'active'
```

## Which APIs should I use to write my queries?

If you are querying a Java application's local cache or querying other members, use [com.gemstone.gemfire.cache.Cache.getQueryService](#).

If you are writing a Java client to server query, use [com.gemstone.gemfire.cache.client.Pool.getQueryService](#).

If you are writing a native client to server query, use the [.NET C# API](#) or the [C++ API](#).

## How do I invoke an object's method in a query?

To use a method in a query, use the attribute name that maps to the public method you want to invoke. For example:

```
/*valid method invocation*/
SELECT DISTINCT * FROM /exampleRegion p WHERE p.positions.size >= 2 - maps to positions.size()
```

## Can I invoke a static method on an object in a query?

No, you cannot invoke a static method on an object. For example, the following query is invalid.

```
/*invalid method invocation*/
SELECT DISTINCT * FROM /exampleRegion WHERE aDay = Day.Wednesday
```

To work around this limitation, write a reusable query that uses a query bind parameter to invoke the static method. Then at query run time, set the parameter to the static method invocation (Day.Wednesday). For example:

```
SELECT DISTINCT * FROM /exampleRegion WHERE aDay = $1
```

## How do I write a reusable query?

Using query APIs, you can set query bind parameters that are passed values at query run time. For example:

```
// Identify your query string.
String queryString = SELECT DISTINCT * FROM /exampleRegion p WHERE p.status = $1;

// Get QueryService from Cache.
QueryService queryService = cache.getQueryService();

// Create the Query Object.
```

```

Query query = queryService.newQuery(queryString);

// Set query parameters.
Object[] params = new Object[1];
params[0] = "active";

// Execute Query locally. Returns results set.
SelectResults results = (SelectResults)query.execute(params);

// Find the Size of the ResultSet.
int size = results.size();

```

If you use a query bind parameter in place of a region path in your path expression, the parameter value must reference a collection (and not a String such as the name of the region path.)

See [Using Query Bind Parameters](#) on page 357 for more details.

### **When should I create indexes to use in my queries?**

Determine whether your query's performance will benefit from an index. For example, in the following query, an index on pkid can speed up the query.

```
SELECT DISTINCT * FROM /exampleRegion portfolio WHERE portfolio.pkid = '123'
```

### **How do I create an indexes?**

Index can be created programmatically using APIs or by using xml. Here are two examples:

#### **Sample Code**

```

QueryService qs = cache.getQueryService();
qs.createIndex("myIndex", "status", "/exampleRegion");
qs.createKeyIndex("myKeyIndex", "id", "exampleRegion");

```

For more information on using this API, see the [GemFire JavaDocs](#).

#### **Sample XML**

```

<region name="portfolios">
<region-attributes . . . >
</region-attributes>
<index name="myIndex">
<functional from-clause="/exampleRegion"
    expression="status"/>
</index>
<index name="myKeyIndex">
<primary-key field="id"/>
</index>
<entry>

```

For more details on indexes, see [Working with Indexes](#) on page 358.

### **Can I create indexes on overflow regions?**

You can create indexes on overflow regions, but you are subject to some limitations. For example, the data contained in the index itself cannot be overflowed to disk. See [Using Indexes with Overflow Regions](#) on page 364 for more information.

## Can I query a partitioned region? Can I perform a join query on a partitioned region?

You can query partitioned regions, but there are some limitations. You cannot perform join queries on partitioned regions, however you can perform equi-join queries on co-located partitioned regions by executing a function on a local data set.

For a full list of restrictions, see [Partitioned Region Query Restrictions](#) on page 367.

## How can I improve the performance of a partitioned region query?

If you know the data you need to query, you can target particular nodes in your queries (thus reducing the number of servers the query needs to access) by executing the query with the FunctionService. See [Querying a Partitioned Region on a Single Node](#) on page 368 for details. If you are querying data that has been partitioned by a key or specific field, you should first create a key index and then execute the query using the FunctionService with the key or field as a filter. See [Optimizing Queries on Data Partitioned by a Key or Field Value](#) on page 370.

## Which query language elements are supported in GemFire?

AND	LIMIT	TO_DATE
AS	LIKE	TYPE
COUNT	NOT	WHERE
DISTINCT	NVL	
ELEMENT	OR	
FROM	ORDER BY	
IMPORT	SELECT	
IN	SET	
IS_DEFINED	<TRACE>	
IS_UNDEFINED	TRUE	

For more information and examples on using each supported keyword, see [Supported Keywords](#) on page 348.

## How do I debug queries?

You can debug a specific query at the query level by adding the <trace> keyword before the query string that you want to debug. Here is an example:

```
<trace> SELECT * FROM /exampleRegion
```

You can also write:

```
<TRACE> SELECT * FROM /exampleRegion
```

When the query is executed, GemFire will log a message in \$GEMFIRE\_DIR/system.log with the following information:

[info 2011/08/29 11:24:35.472 PDT CqServer <main> tid=0x1] Query Executed in 9.619656 ms; rowCount = 99; indexesUsed(0) "select \* from /exampleRegion"

If you want to enable debugging for all queries, you can enable query execution logging by setting a System property on the command line during start-up:

```
gfsh>start server --name=server_name --J=-Dgemfire.Query.VERBOSE=true
```

Or you can set the property programmatically:

```
System.setProperty("gemfire.Query.VERBOSE", "true");
```

## Basic Querying

This section provides a high-level introduction to GemFire querying such as building a query string and describes query language features.

vFabric GemFire provides a SQL-like querying language that allows you to access data stored in GemFire regions. Since GemFire regions are key-value stores where values can range from simple byte arrays to complex nested objects, GemFire uses a query syntax based on OQL (Object Query Language) to query region data. OQL and SQL have many syntactical similarities, however they have significant differences. For example, while OQL does not offer all of the capabilities of SQL like aggregates, OQL does allow you to execute queries on complex object graphs, query object attributes and invoke object methods.

The syntax of a typical GemFire OQL query is:

```
[ IMPORT package ]
SELECT [ DISTINCT ] projectionList
FROM collection1, [ collection2, ... ]
[ WHERE clause ]
[ ORDER BY order_criteria [ desc ] ]
```

Therefore, a simple GemFire OQL query resembles the following:

```
SELECT DISTINCT * FROM /exampleRegion WHERE status = 'active'
```

An important characteristic of GemFire querying to note is that by default, GemFire queries on the values of a region and not on keys. To obtain keys from a region, you must use the keySet path expression on the queried region. For example, `/exampleRegion.keySet`.

For those new to the GemFire querying, see also the [GemFire Querying FAQ and Examples](#) on page 325.

## Advantages of OQL

The following list describes some of the advantages of using an OQL-based querying language:

- You can query on any arbitrary object
- You can navigate object collections
- You can invoke methods and access the behavior of objects
- Data mapping is supported
- You are not required to declare types. Since you do not need type definitions, you can work across multiple languages
- You are not constrained by a schema

## Writing and Executing a Query in vFabric GemFire

The GemFire QueryService provides methods to create the Query object. You can then use the Query object to perform query-related operations.

The QueryService instance you should use depends on whether you are querying the local cache of an application or if you want your application to query the server cache.

### Querying a Local Cache

To query the application's local cache or to query other members, use  
`com.gemstone.gemfire.cache.Cache.getQueryService`.

### Sample Code

```
// Identify your query string.
String queryString = "SELECT DISTINCT * FROM /exampleRegion";
```

```

// Get QueryService from Cache.
QueryService queryService = cache.getQueryService();

// Create the Query Object.
Query query = queryService.newQuery(queryString);

// Execute Query locally. Returns results set.
SelectResults results = (SelectResults)query.execute();

// Find the Size of the ResultSet.
int size = results.size();

// Iterate through your ResultSet.
Portfolio p = (Portfolio)results.iterator().next(); /* Region containing
Portfolio object. */

```

## Querying a Server Cache from a Client

To perform a client to server query, use

```
com.gemstone.gemfire.cache.client.Pool.getQueryService.
```

### Sample Code

```

// Identify your query string.
String queryString = "SELECT DISTINCT * FROM /exampleRegion";

// Get QueryService from client pool.
QueryService queryService = pool.getQueryService();

// Create the Query Object.
Query query = queryService.newQuery(queryString);

// Execute Query locally. Returns results set.
SelectResults results = (SelectResults)query.execute();

// Find the Size of the ResultSet.
int size = results.size();

// Iterate through your ResultSet.
Portfolio p = (Portfolio)results.iterator().next(); /* Region containing
Portfolio object. */

```

Refer to the following JavaDocs for specific APIs:

- [Query package](#)
- [QueryService](#)

## Building a Query String

A query string is a fully formed OQL statement that can be passed to a query engine and executed against a data set. To build a query string, you combine supported keywords, expressions, and operators to create an expression that returns the information you require.

A query string follows the rules specified by the query language and grammar. It can include:

- **Namescopes.** For example, the IMPORT statement. See [IMPORT Statement](#) on page 338.
- **Path expressions.** For example, in the query SELECT \* FROM /exampleRegion, `/exampleRegion` is a path expression. See [FROM Clause](#) on page 338.
- **Attribute names.** For example, in the query SELECT DISTINCT \* FROM /exampleRegion p WHERE p.position1.secId = '1', we access the `secId` attribute of the Position object. See [WHERE Clause](#) on page 339.

- **Method invocations.** For example, in the query `SELECT DISTINCT * FROM /exampleRegion p WHERE p.name.startsWith( 'Bo' )`, we invoke the `startsWith` method on the `Name` object. See [WHERE Clause](#) on page 339.
- **Operators.** For example, comparison operators (`=,<,>,<>`), unary operators (NOT), logical operators (AND, OR) and so on. See [Operators](#) on page 352 for a complete list.
- **Literals.** For example, boolean, date, time and so on. See [Supported Literals](#) on page 354 for a complete list.
- **Query bind parameters.** For example, in the query `SELECT DISTINCT * FROM $1 p WHERE p.status = $2`, `$1` and `$2` are parameters that can be passed to the query during runtime. See [Using Query Bind Parameters](#) on page 357 for more details.
- **Preset query functions.** For example, `ELEMENT(expr)` and `IS_DEFINED(expr)`. See [SELECT Statement](#) on page 345 for other available functions.
- **SELECT statements.** For example, in the example queries above `SELECT *` or `SELECT DISTINCT *`. See [SELECT Statement](#) on page 345 for other available functions.

The components listed above can all be part of the query string, but none of the components are required. At a minimum, a query string contains an expression that can be evaluated against specified data.

The following sections provide guidelines for the query language building blocks that are used when writing typical GemFire queries.

### **IMPORT Statement**

It is sometimes necessary for an OQL query to refer to the class of an object. In cases where the same class name resides in two different namespaces (packages), you must be able to differentiate the classes having the same name.

The **IMPORT** statement is used to establish a name for a class in a query.

```
IMPORT package.Position;
SELECT DISTINCT * FROM /exampleRegion, positions.values positions TYPE
Position WHERE positions.mktValue >= 25.00
```

### **FROM Clause**

Use the **FROM** clause to bring the data you need into scope for the rest of your query. The **FROM** clause also includes object typing and iterator variables.

The query engine resolves names and path expressions according to the name space that is currently in scope in the query.

### **Path Expressions**

The initial name space for any query is composed of:

- **Regions.** In the context of a query, the name of a region is specified by its full path starting with a forward slash (/) and delimited by the forward slash between region names. For example, `/exampleRegion` or `/root/exampleRegion`.
- **Region querying attributes.** From a region path, you can access the Region object's public fields and methods, referred to in querying as the region's attributes. For example, `/exampleRegion.size`.
- **Top-level region data.** You can access entry keys and entry data through the region path.
  1. `/exampleRegion.keySet` returns the Set of entry keys in the region
  2. `/exampleRegion.entrySet` returns the Set of Region.Entry objects
  3. `/exampleRegion.values` returns the Collection of entry values
  4. `/exampleRegion` returns the Collection of entry values

New name spaces are brought into scope based on the **FROM** clause in the **SELECT** statement.

### **Examples:**

Type of Query and Query Description	Query String
Query a region for all distinct values. Returns a collection of unique entry values from the region.	<pre>SELECT DISTINCT * FROM /exampleRegion</pre>
Query the top level region data using entrySet. Returns the keys and positions of Region.Entry objects whose mktValue attribute is greater than 25.00.	<pre>SELECT key, positions FROM /exampleRegion.entrySet, value.positions.values positions WHERE positions.mktValue &gt;= 25.00</pre>
Query the region for its entry values. Returns a set of unique values from Region.Entry objects that have the key equal to 1.	<pre>SELECT DISTINCT entry.value FROM /exampleRegion.entries entry WHERE entry.key = '1'</pre>
Query the region for its entry values. Returns the set of all values from Region.Entry objects that have the key equal to 1.	<pre>SELECT * FROM /exampleRegion.entries entry WHERE entry.value.ID &gt; 1</pre>
Query entry keys in the region. Returns a set of entry keys in the region that have the key equal to '1'.	<pre>SELECT * FROM /exampleRegion.keySet key WHERE key = '1'</pre>
Query values in the region. Returns a collection of entry values in the region that have the status attribute value of 'active'.	<pre>SELECT * FROM /exampleRegion.values portfolio WHERE portfolio.status = 'active'</pre>

## Aliases and Synonyms

In query strings, you can use aliases in path expressions (region and its objects) so that you can refer to the region or objects in other places in the query.

You can also use the **AS** keyword to provide a label for joined path expressions.

### Examples:

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.status = 'active'
```

```
SELECT * FROM /exampleRegion p, p.positions.values AS pos WHERE pos.secId != '1'
```

## Object Typing

Specifying object type in the FROM clause helps the query engine to process the query at optimal speed. Apart from specifying the object types during configuration (using key-constraint and value-constraint) type can be explicitly specified in the query string.

### Example:

```
SELECT DISTINCT * FROM /exampleRegion, positions.values positions TYPE Position WHERE positions.mktValue >= 25.00
```

## WHERE Clause

Each FROM clause expression must resolve to a collection of objects. The collection is then available for iteration in the query expressions that follow in the WHERE clause.

For example:

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.status = 'active'
```

The entry value collection is iterated by the WHERE clause, comparing the status field to the string 'active'. When a match is found, the value object is added to the return set.

In the next example query, the collection specified in the first FROM clause expression is used by the rest of the SELECT statement, including the second FROM clause expression.

```
SELECT DISTINCT * FROM /exampleRegion, positions.values positions WHERE positions.qty > 1000.00
```

### Implementing equals and hashCode Methods

You must implement the `equals` and `hashCode` methods in your custom objects if you are doing ORDER BY and DISTINCT queries on the objects. The methods must conform to the properties and behavior documented in the online Java API documentation for `java.lang.Object`. Inconsistent query results may occur if these methods are absent.

If you have implemented `equals` and `hashCode` methods in your custom objects, you must provide detailed implementations of these methods so that queries execute properly against the objects. For example, assume that you have defined a custom object (`CustomObject`) with the following variables:

```
int ID  
int otherValue
```

Let's put two `CustomObjects` (we'll call them `CustomObjectA` and `CustomObjectB`) into the cache:

`CustomObjectA`:

```
ID=1  
otherValue=1
```

`CustomObjectB`:

```
ID=1  
otherValue=2
```

If you have implemented the `equals` method to simply match on the `ID` field (`ID == ID`), queries will produce unpredictable results.

The following query:

```
SELECT * FROM /CustomObjects c  
WHERE c.ID > 1 AND c.ID < 3  
AND c.otherValue > 0 AND c.otherValue < 3
```

returns two objects, however the objects will be two of either `CustomObjectA` or `CustomObjectB`.

Alternately, the following query:

```
SELECT * FROM /CustomObjects c  
WHERE c.ID > 1 AND c.ID < 3  
AND c.otherValue > 1 AND c.otherValue < 3
```

returns either 0 results or 2 results of `CustomObjectB`, depending on which entry is evaluated last.

To avoid unpredictable querying behavior, implement detailed versions of the `equals` and `hashCode` methods.

### Querying Serialized Objects

Objects must implement `Serializable` if you will be querying partitioned regions or if you are performing client-server querying.

If you are using PDX serialization, you can access the values of individual fields without having to deserialize the entire object. This is accomplished by using PdxInstance, which is a wrapper around the serialized stream. The PdxInstance provides a helper method that takes field-name and returns the value without deserializing the object. While evaluating the query, the query engine will access field values by calling the getField method thus avoiding deserialization.

To use PdxInstances in querying, ensure that PDX serialization reads are enabled in your server's cache. In cache.xml, ensure that the following is set:

```
// Cache configuration setting PDX read behavior
<cache>
  <pdx read-serialized="true">
  ...
  </pdx>
</cache>
```

## Attribute Visibility

You can access any object or object attribute that is available in the current scope of a query. In querying, an object's attribute is any identifier that can be mapped to a public field or method in the object. In the FROM specification, any object that is in scope is valid. Therefore, at the beginning of a query, all locally cached regions and their attributes are in scope.

For attribute Position.secId which is public and has getter method "getSecId()", the query can be written as the following:

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.position1.secId = '1'
SELECT DISTINCT * FROM /exampleRegion p WHERE p.position1.SecId = '1'
SELECT DISTINCT * FROM /exampleRegion p WHERE p.position1.getSecId() = '1'
```

The query engine tries to evaluate the value using the public field value. If a public field value is not found, it makes a get call using field name (note that the first character is uppercase.)

## Joins

If collections in the FROM clause are not related to each other, the WHERE clause can be used to join them.

The statement below returns all portfolios from the /exampleRegion and /exampleRegion2 regions that have the same status.

```
SELECT * FROM /exampleRegion portfolio1, /exampleRegion2 portfolio2 WHERE
portfolio1.status = portfolio2.status
```

To create indexes for region joins you create single-region indexes for both sides of the join condition. These are used during query execution for the join condition. Partitioned regions do not support region joins. For more information on indexes, see [Working with Indexes](#) on page 358.

## Examples:

Query Description	Query String
Query two regions. Return the ID and status for portfolios that have the same status.	<pre>SELECT portfolio1.ID, portfolio2.status FROM /exampleRegion portfolio1, /exampleRegion2 portfolio2 WHERE portfolio1.status = portfolio2.status</pre>
	<pre>SELECT * FROM /exampleRegion portfolio1, portfolio1.positions.values</pre>

Query Description	Query String
	positions1, /exampleRegion2 portfolio2, portfolio2.positions.values positions2 WHERE positions1.secId = positions1.secId
	SELECT * FROM /exampleRegion portfolio1, portfolio1.positions.values positions1, /exampleRegion2 portfolio2, portfolio2.positions.values positions2 WHERE portfolio1.ID = 1 AND positions1.secId = positions1.secID

## LIKE

GemFire offers limited support for the LIKE predicate. LIKE can be used to mean 'equals to'. If you terminate the string with a wildcard (either '%' or '\*'), it behaves like 'starts with'. You can also place a wildcard (either '%' or '\_') at any other position in the comparison string. You can escape the wildcard characters to represent the characters themselves.

You cannot use indexes in queries using LIKE.

### Examples:

Query Description	Query SString
Query the region. Return all objects where status equals 'active'.	SELECT * FROM /exampleRegion p WHERE p.status LIKE 'active'
Query the region using a wild card for comparison. Returns all objects where status begins with 'activ'.	SELECT * FROM /exampleRegion p WHERE p.status LIKE 'activ%'

## Method Invocations

To use a method in a query, use the attribute name that maps to the public method you want to invoke.

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.positions.size >= 2 - maps  
to positions.size()
```

Methods declared to return void evaluate to null when invoked through the query processor.

You cannot invoke a static method. See [Enum Objects](#) on page 343 for more information.

### Methods without parameters

If the attribute name maps to a public method that takes no parameters, just include the method name in the query string as an attribute. For example, emps.isEmpty is equivalent to emps.isEmpty().

In the following example, the query invokes isEmpty on positions, and returns the set of all portfolios with no positions:

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.positions.isEmpty
```

### Methods with parameters

To invoke methods with parameters, include the method name in the query string as an attribute and provide method arguments between parentheses.

This example passes the argument "Bo" to the public method, and returns all names that begin with "Bo".

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.name.startsWith('Bo')
```

For overloaded methods, the query processor decides which method to call by matching the runtime argument types with the parameter types required by the method. If only one method's signature matches the parameters provided, it is invoked. The query processor uses runtime types to match method signatures.

If more than one method can be invoked, the query processor chooses the method whose parameter types are the most specific for the given arguments. For example, if an overloaded method includes versions with the same number of arguments, but one takes a Person type as an argument and the other takes an Employee type, derived from Person, Employee is the more specific object type. If the argument passed to the method is compatible with both types, the query processor uses the method with the Employee parameter type.

The query processor uses the runtime types of the parameters and the receiver to determine the proper method to invoke. Because runtime types are used, an argument with a null value has no typing information, and so can be matched with any object type parameter. When a null argument is used, if the query processor cannot determine the proper method to invoke based on the non-null arguments, it throws an `AmbiguousNameException`.

## Enum Objects

To write a query based on the value of an Enum object field, you must use the `toString` method of the enum object or use a query bind parameter.

For example, the following query is NOT valid:

```
// INVALID QUERY
select distinct * from /QueryRegion0 where aDay = Day.Wednesday
```

The reason it is invalid is that the call to `Day.Wednesday` involves a static class and method invocation which is not supported.

Enum types can be queried by using `toString` method of the enum object or by using bind parameter. When you query using the `toString` method, you must already know the constraint value that you wish to query. In the following first example, the known value is 'active'.

### Examples:

Query Description	Query String
Query enum type using the <code>toString</code> method.	<pre>// eStatus is an enum with values // 'active' and 'inactive' select * from /exampleRegion p where p.eStatus.toString() = 'active'</pre>
Query enum type using a bind parameter. The value of the desired Enum field ( <code>Day.Wednesday</code> ) is passed as an execution parameter.	<pre>select distinct * from /QueryRegion0 where aDay = \$1</pre>

## IN and SET

The IN expression is a boolean indicating if one expression is present inside a collection of expressions of compatible type. The determination is based on the expressions' equals semantics.

If `e1` and `e2` are expressions, `e2` is a collection, and `e1` is an object or a literal whose type is a subtype or the same type as the elements of `e2`, then `e1 IN e2` is an expression of type boolean.

The expression returns:

- TRUE if e1 is not UNDEFINED and is contained in collection e2
- FALSE if e1 is not UNDEFINED and is not contained in collection e2
- UNDEFINED if e1 is UNDEFINED

For example, `2 IN SET(1, 2, 3)` is TRUE.

Another example is when the collection you are querying into is defined by a subquery. This query looks for companies that have an active portfolio on file:

```
SELECT name, address FROM /company
WHERE id IN (SELECT id FROM /portfolios WHERE status = 'active')
```

The interior SELECT statement returns a collection of ids for all /portfolios entries whose status is active. The exterior SELECT iterates over /company, comparing each entry's id with this collection. For each entry, if the IN expression returns TRUE, the associated name and address are added to the outer SELECT's collection.

### Comparing Set Values

The following is an example of a set value type comparison where sp is of type Set:

```
SELECT * FROM /exampleRegion WHERE sp = set('20','21','22')
```

In this case, if sp only contains '20' and '21', then the query will evaluate to false. The query compares the two sets and looks for the presence of all elements in both sets.

For other collections types like list, the query can be written as follows:

```
SELECT * FROM /exampleRegion WHERE sp.containsAll(set('20','21','22'))
```

where sp is of type List.

In order to use it for Set value, the query can be written as:

```
SELECT * FROM /exampleRegion WHERE sp IN SET
(set('20','21','22'),set('10','11','12'))
```

where a set value is searched in collection of set values.

One problem is that you cannot create indexes on Set or List types (collection types) that are not comparable. To workaround this, you can create an index on a custom collection type that implements Comparable.

### Double.NaN and Float.NaN Comparisons

The comparison behavior of Double.NaN and Float.NaN within GemFire queries follow the semantics of the JDK methods `Float.compareTo` and `Double.compareTo`.

In summary, the comparisons differ in the following ways from those performed by the Java language numerical comparison operators (`<`, `<=`, `==`, `>=`, `>`) when applied to primitive double [float] values:

- Double.NaN [Float.NaN] is considered to be equal to itself and greater than all other double [float] values (including Double.POSITIVE\_INFINITY [Float.POSITIVE\_INFINITY]).
- 0.0d [0.0f] is considered by this method to be greater than -0.0d [-0.0f].

Therefore, Double.NaN[Float.NaN] is considered to be larger than

Double.POSITIVE\_INFINITY[Float.POSITIVE\_INFINITY]. Here are some example queries and what to expect.

If p.value is NaN, the following query:	Evaluates to:	Appears in the result set?
<code>SELECT * FROM /positions p WHERE p.value = 0</code>	false	no
<code>SELECT * FROM /positions p WHERE p.value &gt; 0</code>	true	yes

If p.value is NaN, the following query:	Evaluates to:	Appears in the result set?
SELECT * FROM /positions p WHERE p.value >= 0	true	yes
SELECT * FROM /positions p WHERE p.value < 0	false	no
SELECT * FROM /positions p WHERE p.value <= 0	false	no
When p.value and p.value1 are both NaN, the following query:	Evaluates to:	Appears in the result set:
SELECT * FROM /positions p WHERE p.value = p.value1	true	yes

If you combine values when defining the following query in your code, when the query is executed the value itself is considered UNDEFINED when parsed and will not be returned in the result set.

```
String query = "SELECT * FROM /positions p WHERE p.value = " + Float.NaN
```

Executing this query, the value itself is considered UNDEFINED when parsed and will not be returned in the result set.

To retrieve NaN values without having another field already stored as NaN, you can define the following query in your code:

```
String query = "SELECT * FROM /positions p WHERE p.value > " +  
Float.MAX_VALUE;
```

## SELECT Statement

The SELECT statement allows you to filter data from the collection of object(s) returned by a WHERE search operation. The projection list is either specified as \* or as a comma delimited list of expressions.

For \*, the interim results of the WHERE clause are returned from the query.

### Examples:

Query Description	Query String
Query all objects from the region using *. Returns the Collection of portfolios (The exampleRegion contains Portfolio as values).	SELECT * FROM /exampleRegion
Query secIds from positions. Returns the Collection of secIds from the positions of active portfolios.	SELECT secId FROM /exampleRegion, positions.values TYPE Position WHERE status = 'active'
Returns a Collection of struct<type: String, positions: map> for the active portfolios. The second field of the struct is a Map ( jav.util.Map ) object, which contains the positions map as the value.	SELECT "type", positions FROM /exampleRegion WHERE status = 'active'
Returns a Collection of struct<portfolios: Portfolio, values: Position> for the active portfolios.	SELECT * FROM /exampleRegion, positions.values TYPE Position WHERE status = 'active'
Returns a Collection of struct<pflo: Portfolio, posn: Position> for the active portfolios.	SELECT * FROM /exampleRegion portfolio, positions positions TYPE Position WHERE portfolio.status = 'active'

## SELECT Statement Results

The result of a SELECT statement is either UNDEFINED or is a Collection that implements the [SelectResults](#) interface.

The SelectResults returned from the SELECT statement is either:

1. A collection of objects, returned for these two cases:
  - When only one expression is specified by the projection list and that expression is not explicitly specified using the fieldname:expression syntax
  - When the SELECT list is \* and a single collection is specified in the FROM clause
2. A collection of Structs that contains the objects

When a struct is returned, the name of each field in the struct is determined following this order of preference:

1. If a field is specified explicitly using the fieldname:expression syntax, the fieldname is used.
2. If the SELECT projection list is \* and an explicit iterator expression is used in the FROM clause, the iterator variable name is used as the field name.
3. If the field is associated with a region or attribute path, the last attribute name in the path is used.
4. If names cannot be decided based on these rules, arbitrary unique names are generated by the query processor.

## DISTINCT

Use the DISTINCT keyword if you want to limit the results set to unique rows. Note that in the current version of GemFire you are no longer required to use the DISTINCT keyword in your SELECT statement.

```
SELECT DISTINCT * FROM /exampleRegion
```



**Note:** If you are using DISTINCT queries, you must implement the equals and hashCode methods for the objects that you query.

## LIMIT

You can use the LIMIT keyword at the end of the query string to limit the number of values returned.

For example, this query returns at most 10 values:

```
SELECT * FROM /exampleRegion LIMIT 10
```

## ORDER BY

You can order your query results in ascending or descending order by using the ORDER BY clause. You must use DISTINCT when you write ORDER BY queries.

```
SELECT DISTINCT * FROM /exampleRegion WHERE ID < 101 ORDER BY ID
```

The following query sorts the results in ascending order:

```
SELECT DISTINCT * FROM /exampleRegion WHERE ID < 101 ORDER BY ID asc
```

The following query sorts the results in descending order:

```
SELECT DISTINCT * FROM /exampleRegion WHERE ID < 101 ORDER BY ID desc
```



**Note:** If you are using ORDER BY queries, you must implement the equals and hashCode methods for the objects that you query.

## Preset Query Functions

GemFire provides several built-in functions for evaluating or filtering data returned from a query. They include the following:

Function	Description	Example
ELEMENT(expr)	Extracts a single element from a collection or array. This function throws a FunctionDomainException if the argument is not a collection or array with exactly one element.	ELEMENT( SELECT DISTINCT * FROM /exampleRegion WHERE id = 'XYZ-1').status = 'active'
IS_DEFINED(expr)	Returns TRUE if the expression does not evaluate to UNDEFINED.	IS_DEFINED( SELECT DISTINCT * FROM /exampleRegion p WHERE p.status = 'active' )
IS_UNDEFINED (expr)	Returns TRUE if the expression evaluates to UNDEFINED. In most queries, undefined values are not included in the query results. The IS_UNDEFINED function allows undefined values to be included, so you can identify element with undefined values.	SELECT DISTINCT * FROM /exampleRegion p WHERE IS_UNDEFINED(p.status)
NVL(expr1, expr2)	Returns expr2 if expr1 is null. The expressions can be query parameters (bind arguments), path expressions, or literals.	
TO_DATE(date_str, format_str)	Returns a Java Data class object. The arguments must be String S with date_str representing the date and format_str	

Function	Description	Example
	representing the format used by date_str. The format_str you provide is parsed using java.text.SimpleDateFormat.	

## COUNT

The COUNT keyword returns the number of results that match the query selection conditions specified in the WHERE clause. Using COUNT allows you to determine the size of a results set. The COUNT statement always returns an integer as its result.

The following queries are example COUNT queries that return region entries:

```
SELECT COUNT(*) FROM /exampleRegion
SELECT COUNT(*) FROM /exampleRegion WHERE ID > 0
SELECT COUNT(*) FROM /exampleRegion WHERE ID > 0 LIMIT 50
SELECT COUNT(*) FROM /exampleRegion
WHERE ID > 0 AND status LIKE 'act%'
SELECT COUNT(*) FROM /exampleRegion
WHERE ID IN SET(1,2,3,4,5)
```

The following COUNT query returns the total number of StructTypes that match the query's selection criteria.

```
SELECT COUNT(*)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 AND pos.secId = 'IBM'
```

The following COUNT query uses the DISTINCT keyword and eliminates duplicates from the number of results.

```
SELECT DISTINCT COUNT(*)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 OR p.status = 'active' OR pos.secId
OR pos.secId = 'IBM'
```

## Query Language Features

GemFire supports the following high-level querying features.

### Supported Character Sets

GemFire query language supports the full ASCII and Unicode character sets.

### Supported Keywords

Query Language Keyword	Description	Example
AND	Logical operator used to create complex expressions by combining two or more expressions to produce a Boolean result. When you combine two conditional expressions using the AND operator, both conditions must evaluate to true for the entire expression to be true.	See <a href="#">Operators</a>
AS	Used to provide a label for a path expression so you can refer to the path by the label later.	See <a href="#">Aliases and Synonyms</a> on page 339
COUNT	Returns the number of results that match the provided criteria.	See <a href="#">COUNT</a> on page 348
DISTINCT	Restricts the select statement to unique results (eliminates duplicates).	See <a href="#">DISTINCT</a> on page 346

Query Language Keyword	Description	Example
ELEMENT	Query function. Extracts a single element from a collection or array. This function throws a <code>FunctionDomainException</code> if the argument is not a collection or array with exactly one element.	See <a href="#">Preset Query Functions</a> on page 347
FROM	You can access any object or object attribute that is available in the current scope of the query.	See <a href="#">FROM Clause</a> on page 338
IMPORT	Used to establish the namescope for objects.	See <a href="#">IMPORT Statement</a> on page 338
IN	The IN expression is a Boolean indicating whether one expression is present inside a collection of expressions of a compatible type.	See <a href="#">IN and SET</a> on page 343
IS_DEFINED	Query function. Returns TRUE if the expression does not evaluate to UNDEFINED.	See <a href="#">Preset Query Functions</a> on page 347
IS_UNDEFINED	Query function. Returns TRUE if the expression evaluates to UNDEFINED. In most queries, undefined values are not included in the query results. The IS_UNDEFINED function allows undefined values to be included, so you can identify element with undefined values.	See <a href="#">Preset Query Functions</a> on page 347
LIMIT	Limits the number of returned results. If you use the limit keyword, you cannot also run operations on the query result set that perform any kind of summary activities. For example trying to run add or addAll or a SelectResult from a query with a LIMIT clause throws an exception.	See <a href="#">LIMIT</a> on page 347
LIKE	LIKE can be used to mean 'equals to' or if you terminate the string with a wildcard (either '%' or '*'), it behaves like 'starts with'. Note that the wildcard can only be used at the end of the comparison string. You can escape the wildcard characters to represent the characters themselves. You can also use the LIKE predicate if an index is present.	See <a href="#">LIKE</a> on page 342
NOT	The example returns the set of portfolios that have positions. Note that NOT cannot use an index.	See <a href="#">Operators</a>
NVL	Returns expr2 if expr1 is null. The expressions can be query parameters (bind arguments), path expressions, or literals.	See <a href="#">Preset Query Functions</a> on page 347
OR	If an expression uses both AND and OR operators, the AND expression has higher precedence than OR.	See <a href="#">Operators</a>
ORDER BY	Allows you to order query results (either in ascending or descending order).	See <a href="#">ORDER BY</a> on page 347
SELECT	Allows you to filter data from the collection of object(s) returned by a WHERE search operation.	See <a href="#">SELECT Statement</a> on page 345
SET	Specifies a collection of values that can be compared to the returned values of query.	See <a href="#">IN and SET</a> on page 343
<TRACE>	Enables debugging on the following query string.	See <a href="#">Query Debugging</a> on page 373
TO_DATE	Returns a Java Data class object. The arguments must be String S with date_str representing the date and format_str representing the format used by date_str. The format_str you provide is parsed using java.text.SimpleDateFormat.	See <a href="#">Preset Query Functions</a> on page 347
TYPE	Specifying object type in the FROM clause helps the query engine to process the query at optimal speed.	See <a href="#">Object Typing</a> on page 339
WHERE	Resolves to a collection of objects. The collection is then available for iteration in the query expressions that follow in the WHERE clause.	See <a href="#">WHERE Clause</a> on page 339

## Case Sensitivity

Query language keywords such as SELECT, NULL, DATE, and <TRACE> are case-insensitive. Identifiers such as attribute names, method names, and path expressions are case-sensitive.

## Comments in Query Strings

Comment lines begin with - (double dash). Comment blocks begin with /\* and end with \*/. For example:

```
SELECT * --my comment FROM /exampleRegion /* here is
a comment */ WHERE status = 'active'
```

## Query Language Grammar

### Language Grammar

Notation used in the grammar:

<b>n</b>	A nonterminal symbol that has to appear at some place within the grammar on the left side of a rule. All nonterminal symbols have to be derived to be terminal symbols.
<i>t</i>	A terminal symbol (shown in italic bold).
<b>x y</b>	x followed by y
<b>x   y</b>	x or y
<b>(x   y)</b>	x or y
<b>[ x ]</b>	x or empty
<b>{ x }</b>	A possibly empty sequence of x.
<b>comment</b>	descriptive text

Grammar list:

```
symbol ::= expression
query_program ::= [ imports semicolon ] query [semicolon]
imports ::= import { semicolon import }
import ::= IMPORT qualifiedName [ AS identifier ]
query ::= selectExpr | expr
selectExpr ::= SELECT DISTINCT projectionAttributes fromClause [ whereClause ]
projectionAttributes ::= * | projectionList
projectionList ::= projection { comma projection }
projection ::= field | expr [ AS identifier ]
field ::= identifier colon expr
fromClause ::= FROM iteratorDef { comma iteratorDef }
iteratorDef ::= expr [ [ AS ] identifier ] [ TYPE identifier ] | identifier
  IN expr [ TYPE identifier ]
whereClause ::= WHERE expr
expr ::= castExpr
castExpr ::= orExpr | left_paren identifier right_paren castExpr
orExpr ::= andExpr { OR andExpr }
andExpr ::= equalityExpr { AND equalityExpr }
equalityExpr ::= relationalExpr { ( = | <> | != ) relationalExpr }
relationalExpr ::= inExpr { ( < | <= | > | >= ) inExpr }
inExpr ::= unaryExpr { IN unaryExpr }
unaryExpr ::= [ NOT ] unaryExpr
postfixExpr ::= primaryExpr { left_bracket expr right_bracket }
  | primaryExpr { dot identifier [ argList ] }
argList ::= left_paren [ valueList ] right_paren
qualifiedName ::= identifier { dot identifier }
primaryExpr ::= functionExpr
  | identifier [ argList ]
  | undefinedExpr
  | collectionConstruction
```

```

    | queryParam
    | literal
    | ( query )
    | region_path
functionExpr ::= ELEMENT left_paren query right_paren
    | NVL left_paren query comma query right_paren
    | TO_DATE left_paren query right_paren
undefinedExpr ::= IS_UNDEFINED left_paren query right_paren
    | IS_DEFINED left_paren query right_paren
collectionConstruction ::= SET left_paren [ valueList ] right_paren
valueList ::= expr { comma expr }
queryParam ::= $ integerLiteral
region_path ::= forward_slash region_name { forward_slash region_name }
region_name ::= name_character { name_character }
identifier ::= letter { name_character }
literal ::= booleanLiteral
    | integerLiteral
    | longLiteral
    | doubleLiteral
    | floatLiteral
    | charLiteral
    | stringLiteral
    | dateLiteral
    | timeLiteral
    | timestampLiteral
    | NULL
    | UNDEFINED
booleanLiteral ::= TRUE | FALSE
integerLiteral ::= [ dash ] digit { digit }
longLiteral ::= integerLiteral L
floatLiteral ::= [ dash ] digit { digit } dot digit { digit } [ ( E | e )
[ plus | dash ] digit { digit } ] F
doubleLiteral ::= [ dash ] digit { digit } dot digit { digit } [ ( E | e )
[ plus | dash ] digit { digit } ] [ D ]
charLiteral ::= CHAR single_quote character single_quote
stringLiteral ::= single_quote { character } single_quote
dateLiteral ::= DATE single_quote integerLiteral dash integerLiteral dash
integerLiteral single_quote
timeLiteral ::= TIME single_quote integerLiteral colon
    integerLiteral colon integerLiteral single_quote
timestampLiteral ::= TIMESTAMP single_quote
    integerLiteral dash integerLiteral dash integerLiteral integerLiteral
colon
    integerLiteral colon
    digit { digit } [ dot digit { digit } ] single_quote
letter ::= any unicode letter
character ::= any unicode character except 0xFFFF
name_character ::= letter | digit | underscore
digit ::= any unicode digit

```

The expressions in the following are all terminal characters:

```

dot ::= .
left_paren ::= (
right_paren ::= )
left_bracket ::= [
right_bracket ::= ]
single_quote ::= '
underscore ::= _
forward_slash ::= /
comma ::= ,

```

```
semicolon ::= ;
colon ::= :
dash ::= -
plus ::= +
```

## Language Notes

- Query language keywords such as SELECT, NULL, and DATE are case-insensitive. Identifiers such as attribute names, method names, and path expressions are case-sensitive.
- Comment lines begin with -- (double dash).
- Comment blocks begin with /\* and end with \*/.
- String literals are delimited by single-quotes. Embedded single-quotes are doubled.

Examples:

```
'Hello' value = Hello
'He said, ''Hello''' value = He said, 'Hello'
```

- Character literals begin with the CHAR keyword followed by the character in single quotation marks. The single-quotation mark character itself is represented as CHAR "" (with four single quotation marks).
- In the TIMESTAMP literal, there is a maximum of nine digits after the decimal point.

## Operators

vFabric GemFire supports comparison, logical, unary, map, index, dot, and right arrow operators.

### Comparison Operators

Comparison operators compare two values and return the results, either TRUE or FALSE.

The following are supported comparison operators:

= equal to	< less than
$\neq$ not equal to	$\leq$ less than or equal to
$\neq$ not equal to	$\geq$ greater than or equal to

The equal and not equal operators have lower precedence than the other comparison operators. They can be used with null. To perform equality or inequality comparisons with UNDEFINED, use the IS\_DEFINED and IS\_UNDEFINED preset query functions instead of these comparison operators.

### Logical Operators

The logical operators AND and OR allow you to create more complex expressions by combining expressions to produce a boolean result. When you combine two conditional expressions using the AND operator, both conditions must evaluate to true for the entire expression to be true. When you combine two conditional expressions using the OR operator, the expression evaluates to true if either one or both of the conditions are true. You can create complex expressions by combining multiple simple conditional expressions with AND and OR operators. When expressions use AND and OR operators, AND has higher precedence than OR.

### Unary Operators

Unary operators operate on a single value or expression, and have lower precedence than comparison operators in expressions. GemFire supports the unary operator NOT. NOT is the negation operator, which changes the

value of the operand to its opposite. So if an expression evaluates to TRUE, NOT changes it to FALSE. The operand must be a boolean.

## Map and Index Operators

Map and index operators access elements in key/value collections (such as maps and regions) and ordered collections (such as arrays, lists, and Strings). The operator is represented by a set of square brackets "[]" immediately following the name of the collection. The mapping or indexing specification is provided inside these brackets.

Array, list, and String elements are accessed using an index value. Indexing starts from zero for the first element, 1 for the second element and so on. If myList is an array, list, or String and index is an expression that evaluates to a non-negative integer, then myList [ index ] represents the (index+1)th element of myList. The elements of a String are the list of characters that make up the string.

Map and region values are accessed by key using the same syntax. The key can be any Object. For a Region, the map operator performs a non-distributed get in the local cache only - with no use of netSearch. So myRegion[ keyExpression ] is the equivalent of myRegion.getEntry( keyExpression ).getValue.

## Dot, Right Arrow, and Forward Slash Operators

The dot operator '.' separates attribute names in a path expression, and specifies the navigation through object attributes. An alternate equivalent to the dot is the right arrow, "->". The forward slash is used to separate region names when navigating into subregions.

## Reserved Words

### Reserved Words

These words are reserved for the query language and may not be used as identifiers. The words with asterisk (\*) after them are not currently used by GemFire, but are reserved for future implementation.

abs*	dictionary	is_defined	orelse*
all	distinct	is_undefined	query*
and	double	last*	select
andthen*	element	like	set
any*	enum*	limit	short
array	except*	list*	some*
as	exists*	listtaset*	string
asc	false	long	struct*
avg*	first*	map	sum*
bag*	flatten*	max*	time
boolean	float	min*	timestamp
by	for*	mod*	to_date
byte	from	nil	true
char	group*	not	type
collection	having*	null	undefine*
count	import	nvl	undefined
date	in	octet	union*
declare*	int	or	unique*
define*	intersect*	order	where
desc	interval*		

To access any method, attribute, or named object that has the same name as a query language reserved word, enclose the name within double quotation marks.

Examples:

```
SELECT DISTINCT "type" FROM /portfolios WHERE status = 'active'  
SELECT DISTINCT * FROM /region1 WHERE emps."select"() < 100000
```

## Supported Literals

vFabric GemFire supports the following literal types:

- **boolean.** A boolean value, either TRUE or FALSE
- **int and long.** An integer literal is of type long if has a suffix of the ASCII letter L. Otherwise it is of type int.
- **floating point.** A floating-point literal is of type float if it has a suffix of an ASCII letter F. Otherwise its type is double. Optionally, it can have a suffix of an ASCII letter D. A double or floating point literal can optionally include an exponent suffix of E or e, followed by a signed or unsigned number.
- **string.** String literals are delimited by single quotation marks. Embedded single-quotation marks are doubled. For example, the character string 'Hello' evaluates to the value Hello, while the character string 'He said, "Hello"' evaluates to He said, 'Hello'. Embedded newlines are kept as part of the string literal.
- **char.** A literal is of type char if it is a string literal prefixed by the keyword CHAR, otherwise it is of type string. The CHAR literal for the single-quotation mark character is CHAR "" (four single quotation marks).
- **date.** A java.sql.Date object that uses the JDBC format prefixed with the DATE keyword: DATE yyyy-mm-dd. In the Date, yyyy represents the year, mm represents the month, and dd represents the day. The year must be represented by four digits; a two-digit shorthand for the year is not allowed.
- **time.** A java.sql.Time object that uses the JDBC format (based on a 24-hour clock) prefixed with the TIME keyword: TIME hh:mm:ss. In the Time, hh represents the hours, mm represents the minutes, and ss represents the seconds.
- **timestamp.** A java.sql.Timestamp object that uses the JDBC format with a TIMESTAMP prefix: TIMESTAMP yyyy-mm-dd hh:mm:ss.fffffffff. In the Timestamp, yyyy-mm-dd represents the date, hh:mm:ss represents the time, and ffffffffff represents the fractional seconds (up to nine digits).
- **NIL.** Equivalent alternative of NULL.
- **NULL.** The same as null in Java.
- **UNDEFINED.** A special literal that is a valid value for any data type. An UNDEFINED value is the result of accessing an attribute of a null-valued attribute. Note that if you access an attribute that has an explicit value of null, then it is not undefined. For example if a query accesses the attribute address.city and address is null, the result is undefined. If the query accesses address, then the result is not undefined, it is NULL.

## Comparing Values With java.util.Date

You can compare temporal literal values DATE, TIME, and TIMESTAMP with java.util.Date values. There is no literal for java.util.Date in the query language.

## Type Conversion

The GemFire query processor performs implicit type conversions and promotions under certain cases in order to evaluate expressions that contain different types. The query processor performs binary numeric promotion, method invocation conversion, and temporal type conversion.

## Binary Numeric Promotion

The query processor performs binary numeric promotion on the operands of the following operators:

- Comparison operators <, <=, >, and >=
- Equality operators = and <>

- Binary numeric promotion widens the operands in a numeric expression to the widest representation used by any of the operands. In each expression, the query processor applies the following rules in the prescribed order until a conversion is made:
  1. If either operand is of type double, the other is converted to double
  2. If either operand is of type float, the other is converted to float
  3. If either operand is of type long, the other is converted to long
  4. Both operands are converted to type int char

### **Method Invocation Conversion**

Method invocation conversion in the query language follows the same rules as Java method invocation conversion, except that the query language uses runtime types instead of compile time types, and handles null arguments differently than in Java. One aspect of using runtime types is that an argument with a null value has no typing information, and so can be matched with any type parameter. When a null argument is used, if the query processor cannot determine the proper method to invoke based on the non-null arguments, it throws an `AmbiguousNameException`

### **Temporal Type Conversion**

The temporal types that the query language supports include the Java types `java.util.Date` , `java.sql.Date` , `java.sql.Time` , and `java.sql.Timestamp` , which are all treated the same and can be compared and used in indexes. When compared with each other, these types are all treated as nanosecond quantities.

### **Enum Conversion**

Enums are not automatically converted. To use Enum values in query, you must use the `toString` method of the enum object or use a query bind parameter. See [Enum Objects](#) on page 343 for more information.

### **Query Evaluation of `Float.NaN` and `Double.NaN`**

`Float.NaN` and `Double.NaN` are not evaluated as primitives; instead, they are compared in the same manner used as the JDK methods `Float.compareTo` and `Double.compareTo`. See [Double.NaN and Float.NaN Comparisons](#) on page 344 for more information.

### **Query Language Restrictions and Unsupported Features**

At a high level, GemFire does not support the following querying features:

- `GROUPBY` is not supported.
- Indexes targeted for joins across more than one region are not supported
- Static method invocations. For example, the following query is invalid:
 

```
SELECT DISTINCT * FROM /QueryRegion0 WHERE aDay = Day.Wednesday
```
- Aggregate functions are not supported.
- You cannot use indexes in queries that use the `LIKE` predicate.
- You cannot create an index on fields using Set/List types (Collection types) that are not comparable. The OQL index implementation expects fields to be Comparable. To workaround this, you can create a custom Collection type that implements Comparable.
- `ORDER BY` is only supported with `DISTINCT` queries.

In addition, there are some specific limitations on partitioned region querying. See [Partitioned Region Query Restrictions](#) on page 367.

## Advanced Querying

This section includes advanced querying topics such as using query indexes, using query bind parameters, querying partitioned regions and query debugging.

### Performance Considerations

Some general performance tips:

- Improve query performance whenever possible by creating indexes. See [Tips and Guidelines on Using Indexes](#) on page 358 for some scenarios for using indexes.
- Use bind parameters for frequently used queries. When you use a bind parameter, the query is compiled once. This improves the subsequent performance of the query when it is re-run. See [Using Query Bind Parameters](#) on page 357 for more details.
- When querying partitioned regions, execute the query using the FunctionService. This function allows you to target a particular node, which will improve performance greatly by avoiding query distribution. See [Querying a Partitioned Region on a Single Node](#) on page 368 for more information.
- Use key indexes when querying data that has been partitioned by a key or field value. See [Optimizing Queries on Data Partitioned by a Key or Field Value](#) on page 370.
- The size of a query result set depends on the restrictiveness of the query and the size of the total data set. A partitioned region can hold much more data than other types of regions, so there is more potential for larger result sets on partitioned region queries. This could cause the member receiving the results to run out of memory if the result set is very large.

### Monitoring Queries for Low Memory

The query monitoring feature prevents out-of-memory exceptions from occurring when you execute queries or create indexes.

This feature is automatically enabled when you set a `critical-heap-percentage` attribute for the `resource-manager` element in `cache.xml` or by using the `cache.getResourceManager().setCriticalHelpPercentage(float heapPercentage)` API. Use this feature to cancel out queries that are taking too long and to warn the user that there are low memory conditions when they are running queries or creating indexes.

You can override this feature by setting the system property `gemfire.cache.DISABLE_QUERY_MONITOR_FOR_LOW_MEMORY` to true.

When the query memory monitoring feature is on, the default query time out is set to five hours. You can override this value by setting a larger or smaller, non -1 value to the existing query time out system variable `gemfire.cache.MAX_QUERY_EXECUTION_TIME`.

When system memory is low (as determined by the critical heap percentage threshold that you defined in `cache.xml` or in the `getResourceManager` API ), queries will throw a `QueryExecutionLowMemoryException`. Any indexes that are in the process of being created will throw an `InvalidIndexException` with the message indicating the reason.

### Partitioned Region Queries and Low Memory

Partitioned region queries are likely causes for out-of-memory exceptions. If query monitoring is enabled, partitioned region queries drop or ignore results that are being gathered by other servers if the executing server is low in memory.

Query-monitoring does not address a scenario in which a low-level collection is expanded while the partitioned region query is gathering results. For example, if a row is added and then causes a Java level collection or array to expand, it is possible to then encounter an out-of-memory exception. This scenario is rare and is only possible if the collection size itself expands before a low memory condition is met and then expands beyond the remaining

available memory. As a workaround, in the event that you encounter this situation, you may be able to tune the system by additionally lowering the `critical-heap-percentage`.

## Using Query Bind Parameters

Using query bind parameters in GemFire queries is similar to using prepared statements in SQL where parameters can be set during query execution. This allows user to build a query once and execute it multiple times by passing the query conditions during run time.

The use of query bind parameters is now supported in Client-to-Server queries.

The query parameters are identified by a dollar sign, \$ , followed by a digit that represents the parameter's position in the parameter array passed to the execute method. Counting begins at 1, so \$1 references the first bound attribute, \$2 the second attribute, and so on.

The Query interface provides an overloaded execute method that accepts parameters inside an Object array. See the [Query.execute JavaDocs](#) for more details.

The 0th element of the Object array is used for the first query parameter, and so on. If the parameter count or parameter types do not match the query specification, the execute method throws an exception. Specifically, if you pass in the wrong number of parameters, the method call throws a `QueryParameterCountInvalidException`. If a parameter object type is not compatible with what is expected, the method call throws a `TypeMismatchException`.

In the following example, the first parameter, the integer **2**, is bound to the first element in the object array. The second parameter, **active**, is bound to the second element.

### Sample Code

```
// Identify your query string.
String queryString = SELECT DISTINCT * FROM /exampleRegion p WHERE p.status
= $1;

// Get QueryService from Cache.
QueryService queryService = cache.getQueryService();

// Create the Query Object.
Query query = queryService.newQuery(queryString);

// Set query parameters.
Object[] params = new Object[1];
params[0] = "active";

// Execute Query locally. Returns results set.
SelectResults results = (SelectResults)query.execute(params);

// Find the Size of the ResultSet.
int size = results.size();
```

## Using Query Bind Parameters in the Path Expression

Additionally the query engine supports the use of query bind parameter in place of a region path. When you specify a bind parameter in the query's FROM clause, the parameter's referenced value must be bound to a collection.

For example, the following query can be used on any collection by passing in the collection as a query parameter value. In this query you could pass in a Region Object for \$1, but not the String name of a region.

```
SELECT DISTINCT * FROM $1 p WHERE p.status = $2
```

## Working with Indexes

The GemFire Enterprise query engine supports indexing. An index can provide significant performance gains for query execution. A query run without the aid of an index iterates through every object in the collection. If an index is available that matches part or all of the query specification, the query iterates only over the indexed set, and query processing time can be reduced.

### Tips and Guidelines on Using Indexes

Optimizing your queries with indexes requires a cycle of careful planning, testing, and tuning. Poorly-defined indexes can degrade the performance of your queries instead of improving it. This section gives guidelines for index usage in the query service.

When creating indexes, keep in mind the following:

- Indexes incur maintenance costs as they must be updated when the indexed data changes. An index that requires many updates and is not used very often may require more system resources than using no index at all.
- Indexes consume memory.
- Indexes have limited support on overflow regions. See [Using Indexes with Overflow Regions](#) on page 364 for details.

### Tips for Writing Queries that Use Indexes

As with query processors that run against relational databases, the way a query is written can greatly affect execution performance. Among other things, whether indexes are used depends on how each query is stated. These are some of the things to consider when optimizing your GemFire queries for performance:

- In general an index will improve query performance if the FROM clauses of the query and index match exactly.
- The query evaluation engine does not have a sophisticated cost-based optimizer. It has a simple optimizer which selects best index (one) or multiple indexes based on the index size and the operator that is being evaluated.
- For AND operators, you may get better results if the conditions that use indexes and conditions that are more selective come before other conditions in the query.
- Indexes are not used in expressions that contain NOT, so in a WHERE clause of a query, `qty >= 10` could have an index on `qty` applied for efficiency. However, `NOT (qty < 10)` could not have the same index applied.

## Creating, Listing and Removing Indexes

The GemFire QueryService API provides methods to create, list and remove the index. You can also use gfsh command-line interface to create, list and remove indexes, and use cache.xml to create an index.

### Creating Indexes

Indexes can be created programmatically, by using the gfsh command line interface or by using cache.xml.

To create an index, use one of the following QueryService methods:

- `createIndex`. Creates the default type of index, a range index. Use this type of index if you will be writing queries that will be doing any kind of comparison operation besides an equality comparison.
- `createKeyIndex`. Creates a key index. See [Creating Key Indexes](#) on page 360 for more information.
- `createHashIndex`. Creates a hash index. See [Creating Hash Indexes](#) on page 361 for more information.

The following sections provide examples of index creation:

#### Using Java API:

```
QueryService qs = cache.getQueryService();
qs.createIndex("myIndex", "status", "/exampleRegion");
```

```
qs.createKeyIndex( "myKeyIndex", "id", "/exampleRegion");
qs.createHashIndex( "myHashIndex", "mktValue", "/exampleRegion");
```

#### Using gfsh:

```
gfsh> create index --name=myIndex --expression=status --region=/exampleRegion
gfsh> create index --name=myKeyIndex --expression=id --region=/exampleRegion
gfsh> create index --name=myHashIndex --expression=mktValue
--region=/exampleRegion
```

See [Index Commands](#) on page 668 for more examples.

#### Using cache.xml:

```
<region name=exampleRegion>
<region-attributes . . . >
</region-attributes>
<index name="myIndex" from-clause="/exampleRegion" expression="status"/>
<index name="myKeyIndex" from-clause="/exampleRegion" expression="id"
key-index="true"/>
<index name="myHashIndex" from-clause="/exampleRegion p"
expression="p.mktValue" type="hash"/>
...
</region>
```



**Note:** If you do not specify the type of index in cache.xml, the type defaults to "range".

### Listing Indexes

To retrieve a list of indexes from the cache or region, use the `QueryService.getIndexes` method or the `gfsh` command line interface.

#### Using Java API:

```
QueryService qs = cache.getQueryService();
qs.getIndexes(); //returns a collection of all indexes in the cache
qs.getIndexes(exampleRegion); //returns a collection of all indexes in
exampleRegion
qs.getIndexes(exampleRegion, myKeyIndex); //returns the index named
myKeyIndex from the exampleRegion
```

#### Using gfsh:

```
gfsh> list indexes
gfsh> list indexes --with-stats
```

### Removing Indexes

To remove an index or all indexes from the cache or region, use the `QueryService.removeIndexes` method or the `gfsh` command line interface.

#### Using Java API:

```
QueryService qs = cache.getQueryService();
qs.removeIndexes(); //removes all indexes from the cache
qs.removeIndexes(myKeyIndex); //removes the index named myKeyIndex
```

```
qs.removeIndexes(exampleRegion); //removes all indexes from the exampleRegion
```

#### Using gfsh:

```
gfsh> destroy index
gfsh> destroy index --name=myIndex
gfsh> destroy index --region=/exampleRegion
```

#### Creating Key Indexes

You can create key indexes by using the `createKeyIndex` method of the `QueryService` or by defining the index in `cache.xml`. Creating a key index makes the query service aware of the relationship between the values in the region and the keys in the region.

The `FROM` clause for a primary key index must be just a region path. The indexed expression is an expression that, when applied to an entry value, produces the key. For example, if a region has `Portfolios` as the values and the keys are the `id` field of the `Portfolios` region, the indexed expression is `id`.

Creating a key index is a good way to improve query performance when data is partitioned using a key or a field value. You can then use the `FunctionService` (using the partitioned key as a filter passed to the function and as part of the query equality condition) to execute the query against the indexed data. See [Optimizing Queries on Data Partitioned by a Key or Field Value](#) on page 370 for more details.

There are two issues to note with key indexes:

- The key index is not sorted. Without sorting, you can only do equality tests. Other comparisons are not possible. To obtain a sorted index on your primary keys, create a functional index on the attribute used as the primary key.
- The query service is not automatically aware of the relationship between the region values and keys. For this, you must create the key index.



**Note:** Using a key-index with an explicit type='range' in the `cache.xml` will lead to an exception. Key indexes will not be used in 'range' queries.

#### Examples of Creating a Key Index

##### Using Java API:

```
QueryService qs = cache.getQueryService();
qs.createKeyIndex("myKeyIndex", "id", "/exampleRegion");
```

##### Using gfsh:

```
gfsh> create index --name=myKeyIndex --expression=id --region=/exampleRegion
```

##### Using `cache.xml`:

```
<region name=exampleRegion>
<region-attributes . . . >
</region-attributes>
<index name="myKeyIndex" from-clause="/exampleRegion" expression="id"
key-index="true" />
...
</region>
```



**Note:** If you do not specify the type of index when defining indexes using `cache.xml`, the type defaults to "range".

## Creating Hash Indexes

### Why Create a HashIndex

By creating a hash index, you can vastly improve the memory usage of your index. If you are doing equality based queries, your indexing footprint is significantly reduced. As an example, previously when you created an index that contained string fields, copies of the strings were included in the index. By using hash indexes, we ensure that indexed expressions are canonicalized and stored in the index as pointers to the PDX objects lying in the region. In internal test cases we have seen improvements in the 30% range depending on the keys and data being used.



**Note:** Your performance results may vary depending on the keys and data being indexed.

### Performance Considerations

The performance of put operations when using a hash index should be comparable to other indexes or slightly slower. Queries themselves are expected to be slightly slower due to the implementation of hash index and the cost of recalculating the key on request, which is the trade-off for the space savings that using a hash index provides.

### Limitations

The following limitations must be considered when creating hash indexes:

- You can only use hash indexes with equals and not equals queries.
- Hash index maintenance will be slower than the other indexes due to synchronized add methods.
- Hash indexes cannot be maintained asynchronously. If you attempt to create a hash index on a region with asynchronous set as the maintenance mode, an exception will be thrown.

### Examples of Creating a Hash Index

#### Using the Java API:

```
QueryService qs = cache.getQueryService();
qs.createHashIndex("myHashIndex", "mktValue", "/exampleRegion");
```

#### Using gfsh:

```
gfsh> create index --name=myHashIndex --expression=mktValue
--region=/exampleRegion
```

#### Using cache.xml:

```
<region name=exampleRegion>
<region-attributes . . . >
</region-attributes>
<index name="myHashIndex" from-clause="/exampleRegion p"
expression="p.mktValue" type="hash"/>
. .
</region>
```

### Maintaining Indexes (Synchronously or Asynchronously) and Index Storage

#### Index Maintenance Behavior

Indexes are automatically kept current with the region data they reference. The region attribute `IndexMaintenanceSynchronous` specifies whether the region indexes are updated synchronously when a region is modified or asynchronously in a background thread. Asynchronous index maintenance batches up

multiple updates to the same region key. The default mode is synchronous, since this provides the greatest consistency with region data.

See [AttributesFactory.setIndexMaintenanceSynchronous](#).

This declarative index creation sets the maintenance mode to asynchronous:

```
<region-attributes index-update-type="asynchronous">
</region-attributes>
```

## Internal Index Structure and Storage

Indexes are stored either as compact or non-compact data structures based on the indexed expression (even if the index key type is the same.) For example, consider the following Passenger object:

```
Passenger {
    String name,
    Date travelDate,
    int age,
    Flight flt,
}
Flight {
    int flightId,
    String origin,
    String dest,
}
```

An index on the Passenger name field will have different memory space requirements in the cache than the Flight origin field even though they are both String field types. The internal data structure selected by GemFire for index storage will depend on the field's level in the object. In this example, name is a top-level field and an index on name can be stored as a compact index. Since origin is a second-level field, any index that uses origin as the indexed expression will be stored as a non-compact index.

### Compact Index

A compact index has simple data structures to minimize its footprint, at the expense of doing extra work at index maintenance. This index does not support the storage of projection attributes.

Currently compact indexes are only selected only supports the creation of an index on a region path. In addition, the following conditions must be met:

- Index maintenance is synchronous.
- The indexed expression is a path expression.
- The FROM clause has only one iterator. This implies that there is only one value in the index for each region entry and it is directly on the region values (not supported with keys, entries).

### Non-Compact Index

Used whenever a compact index cannot be used.

### Using Indexes on Single Region Queries

Queries with one comparison operation may be improved with either a primary key or functional key index, depending on whether the attribute being compared is also the primary key.

```
SELECT DISTINCT * FROM /exampleRegion portfolio WHERE portfolio.pkid = '123'
```

If pkid is the key in the /exampleRegion region, creating a primary key index on pkid is the best choice as a primary key index does not have maintenance overhead. If pkid is not the key, a functional index on pkid should improve performance.

With multiple comparison operations, you can create a functional index on one or more of the attributes. Try the following:

1. Create a single index on the condition you expect to have the smallest result set size. Check performance with this index.
2. Keeping the first index, add an index on a second condition. Adding the second index may degrade performance. If it does, remove it and keep only the first index. The order of the two comparisons in the query can also impact performance. Generally speaking, in OQL queries, as in SQL queries, you should order your comparisons so the earlier ones give you the fewest results on which to run subsequent comparisons.

For this query, you would try a functional index on name, age, or on both:

```
SELECT DISTINCT * FROM /exampleRegion portfolio
WHERE portfolio.status = 'active' AND portfolio.ID > 45
```

For queries with nested levels, you may get better performance by drilling into the lower levels in the index as well as in the query.

This query drills down one level:

```
SELECT DISTINCT * FROM /exampleRegion portfolio, portfolio.positions.values
positions
WHERE positions.secId = 'AOL' and positions.MktValue > 1
```

### Using Indexes with Equi-Join Queries

Equi-join queries are queries in which two regions are joined through an equality condition in the WHERE clause.

1. Create an index for each side of the equi-join condition. The query engine can quickly evaluate the query's equi-join condition by iterating over the keys of the left-side and right-side indexes for an equality match.



**Note:** Equi-join queries require regular indexes. Key indexes are not applied to equi-join queries.

For this query:

```
SELECT DISTINCT inv.name, ord.orderID, ord.status
FROM /investors inv, /orders ord
WHERE inv.investorID = ord.investorID
```

Create two indexes:

FROM clause	Indexed expression
/investors inv	inv.investorID
/orders ord	ord.investorID

2. If there are additional, single-region queries in a query with an equi-join condition, create additional indexes for the single-region conditions only if you are able to create at least one such index for each region in the query. Any indexing on a subset of the regions in the query will degrade performance.

For this example query:

```
SELECT DISTINCT *
FROM /investors inv, /securities sc, inv.heldSecurities inv_hs
    WHERE sc.status = "active"
        AND inv.name = "xyz"
        AND inv.age > 75
        AND inv_hs.secName = sc.secName
```

Create the indexes for the equi-join condition:

FROM clause	Indexed expression
/investors inv, inv.heldSecurities inv_hs	inv_hs.secName

FROM clause	Indexed expression
/securities sc	sc.secName

Then, if you create any more indexes, create one on sc.status and one on inv.age or inv.name or both.

### Using Indexes with Overflow Regions

You can use indexes for querying on overflow regions with these caveats:

- You must use synchronous index maintenance for the region. This is the default maintenance setting.
- The index from clause must specify only one iterator, and it must refer to the keys or entry values. The index cannot refer to the region's entrySet.
- The index data itself is not stored overflowed to disk .

#### Examples:

The following example index creation calls DO NOT work for overflow regions.

```
// This index will not work on an overflow region because there are two
// iterators in the FROM clause.
createIndex("secIdIndex", "b.secId", "/portfolios pf, pf.positions.values
b");
// This index will not work on an overflow region because the FROM clause
// specifies the entrySet
createIndex("idx1", "entries.value.getID", "/exampleRegion.entrySet()
entries");
```

The following example indexes will work for overflow regions.

```
createIndex("pkidIndex", "p.pkid", "/Portfolios p");
createIndex("idx1", "ks.toString", "/portfolio.keys() ks");
```

### Using Indexes on Equi-Join Queries using Multiple Regions

To query across multiple regions, identify all equi-join conditions. Then, create as few indexes for the equi-join conditions as you can while still joining all regions. If there are equi-join conditions that redundantly join two regions - in order to more-finely filter the data, for example - creating redundant indexes for these joins will negatively impact performance. Create indexes only on one equi-join condition for each region pair.

In this example query:

```
SELECT DISTINCT *
FROM /investors inv, /securities sc, /orders or,
inv.ordersPlaced inv_op, or.securities or_sec
    WHERE inv_op.orderID = or.orderID
        AND or_sec.secID = sc.secID
```

All conditions are required to join the regions, so you would create four indexes, two for each equi-join condition:

FROM clause	Indexed expression
/investors inv, inv.ordersPlaced inv_op	inv_op.orderID
/orders or, or.securities or_sec	or.orderID
FROM clause	Indexed expression
/orders or, or.securities or_sec	or_sec.secID
/securities sc	sc.secID

Adding another condition to the example:

```
SELECT DISTINCT *
FROM /investors inv, /securities sc, /orders or,
inv.ordersPlaced inv_op, or.securities or_sec, sc.investors sc_invs
WHERE inv_op.orderID = or.orderID
AND or_sec.secID = sc.secID
AND inv.investorID = sc_invs.investorID
```

You would still only want to use four indexes in all, as that's all you need to join all of the regions. You would need to choose the most performant two of the following three index pairs:

FROM clause	Indexed expression
/investors inv, inv.ordersPlaced inv_op	inv_op.orderID
/orders or, or.securities or_sec	or.orderID

FROM clause	Indexed expression
/orders or, or.securities or_sec	or_sec.secID
/securities sc, sc.investors sc_invs	sc.secID

FROM clause	Indexed expression
/investors inv, inv.ordersPlaced inv_op	inv.investorID
/securities sc, sc.investors sc_invs	sc_invs.investorID

The most performant set is that which narrows the data to the smallest result set possible. Examine your data and experiment with the three index pairs to see which provides the best performance.

## Index Samples

The following are some code samples for creating query indexes.

```
// Key index samples. The field doesn't have to be present.
createKeyIndex("pkidIndex", "p.pkid1", "/root/exampleRegion p");

createKeyIndex("Index4", "ID", "/portfolios");

// Simple index
createIndex("pkidIndex", "p.pkid", "/root/exampleRegion p");
createIndex("i", "p.status", "/exampleRegion p")
createIndex("i", "p.ID", "/exampleRegion p")
createIndex("i", "p.position1.secId", "/exampleRegion p"

// On Set type
createIndex("setIndex", "s", "/root/exampleRegion p, p.sp s");

// Positions is a map
createIndex("secIdIndex", "b.secId", "/portfolios pf, pf.positions.values b");

//...
createIndex("i", "pf.collectionHolderMap[(pf.Id).toString()].arr[pf.ID]", "/exampleRegion pf")
createIndex("i", "pf.ID", "/exampleRegion pf", "pf.positions.values pos")
createIndex("i", "pos.secId", "/exampleRegion pf", "pf.positions.values pos")
createIndex("i", "e.value.getID()", "/exampleRegion.entrySet e")
```

```

createIndex("i", "e.value.ID", "/exampleRegion.entrySet e")

//...
createIndex("i", "entries.value.getID", "/exampleRegion.entrySet() entries")
createIndex("i", "ks.toString", "/exampleRegion.getKeys() ks")
createIndex("i", "key.status", "/exampleRegion.keys key")
createIndex("i", "secIds.length", "/exampleRegion p, p.secIds secIds")
createIndex("i", "secId", "/portfolios.asList[1].positions.values")
createIndex("i", "secId", "/portfolios['1'].positions.values")

//Index on Map types
createIndex("i", "p.positions['key1']", "/exampleRegion p")
createIndex("i", "p.positions['key1','key2','key3','key7']", "/exampleRegion p")
createIndex("i", "p.positions[*]", "/exampleRegion p")

```

The following are some sample queries on indexes.

```

SELECT * FROM (SELECT * FROM /R2 m) r2, (SELECT * FROM /exampleRegion e
WHERE e.pkid IN r2.sp) p

SELECT * FROM (SELECT * FROM /R2 m WHERE m.ID IN SET (1, 5, 10)) r2,
(SELECT * FROM /exampleRegion e WHERE e.pkid IN r2.sp) p

//examples using position index in the collection
SELECT * FROM /exampleRegion p WHERE p.names[0] = 'aaa'

SELECT * FROM /exampleRegion p WHERE p.position3[1].portfolioId = 2

SELECT DISTINCT positions.values.toArray[0], positions.values.toArray[0],
status
FROM /exampleRegion

```

## Querying Partitioned Regions

GemFire allows you to manage and store large amounts of data across distributed nodes using partitioned regions. The basic unit of storage for a partitioned region is a bucket, which resides on a GemFire node and contains all the entries that map to a single hashcode. In a typical partitioned region query, the system distributes the query to all buckets across all nodes, then merges the result sets and sends back the query results.

The following list summarizes the querying functionality supported by GemFire for partitioned regions:

- **Ability to target specific nodes in a query.** If you know that a specific bucket contains the data that you want to query, you can use a function to ensure that your query only runs the specific node that holds the data. This can greatly improve query efficiency. The ability to query data on a specific node is only available if you are using functions and if the function is executed on one single region. In order to do this, you need to use `Query.execute(RegionFunctionContext context)`. See the [Java API](#) and [Querying a Partitioned Region on a Single Node](#) on page 368 for more details.
- **Ability to optimize partitioned region query performance using key indexes.** You can improve query performance on data that is partitioned by key or a field value by creating a key index and then executing the query using `use Query.execute(RegionFunctionContext context)` with the key or field value used as filter. See the [Java API](#) and [Optimizing Queries on Data Partitioned by a Key or Field Value](#) on page 370 for more details.
- **Ability to perform equi-join queries between partitioned regions and between partitioned regions and replicated regions.** Join queries between partitioned region and between partitioned regions and replicated regions are supported through the function service. In order to perform equi-join operations on partitioned regions or partitioned regions and replicated regions, the partitioned regions must be co-located, and you need

to use the need to use `Query.execute(RegionFunctionContext context)`. See the [Java API](#) and [Performing an Equi-Join Query on Partitioned Regions](#) on page 371 for more details.

## Partitioned Region Query Restrictions

### Query Restrictions in Partitioned Regions

Partitioned region queries function the same as non-partitioned region queries, except for the restrictions listed in this section. Partitioned region queries that do not follow these guidelines generate an `UnsupportedOperationException`.

- Join queries between partitioned region and between partitioned regions and replicated regions are supported through the function service only. Join queries on partitioned regions are not supported through the client server API.
- You can run join queries on partitioned regions and on partitioned regions and replicated regions only if they are co-located. Equi-join queries are supported only on partitioned regions that are co-located and where the co-located columns are indicated in the WHERE clause of the query. In the case of multi-column partitioning, there should also be an AND clause in the WHERE specification. See [Co-locate Data from Different Partitioned Regions](#) on page 223 for more information on partitioned region co-location.
- Equi-join queries are allowed between partitioned regions and between partitioned regions and local replicated regions as long as the local replicated region also exists on all partitioned region nodes. To perform a join query on a partitioned region and another region (partitioned or not), you need to use the `query.execute` method and supply it with a function execution context. See [Performing an Equi-Join Query on Partitioned Regions](#) on page 371 for an example.
- The query must be just a SELECT expression (as opposed to arbitrary OQL expressions), preceded by zero or more IMPORT statements. For example, this query is not allowed because it is not just a SELECT expression:

```
// NOT VALID for partitioned regions
(SELECT DISTINCT *FROM /prRgn WHERE attribute > 10).size
```

This query is allowed:

```
// VALID for partitioned regions
SELECT DISTINCT *FROM /prRgn WHERE attribute > 10
```

- The SELECT expression itself can be arbitrarily complex, including nested SELECT expressions, as long as only one partitioned region is referenced.
- The partitioned region reference can only be in the first FROM clause iterator. Additional FROM clause iterators are allowed if they do not reference any regions (such as drilling down into the values in the partitioned region).
- The first FROM clause iterator must contain only one reference to the partitioned region (the reference can be a parameter, such as \$1).
- The first FROM clause iterator cannot contain a subquery, but subqueries are allowed in additional FROM clause iterators.
- You can use ORDER BY on partitioned region queries, but the fields that are specified in the ORDER BY clause must be part of the projection list.

### Using ORDER BY on Partitioned Regions

To execute a query with an ORDER BY clause on a partitioned region, the fields specified in the ORDER BY clause must be part of the projection list.

When an ORDER BY clause is used with a partition region query, the query is executed separately on each region host, the local query coordinator, and all remote members. The results are all gathered by the query coordinator. The cumulative result set is built by applying ORDER BY on the gathered results. If the LIMIT clause is also used in the query, ORDER BY and LIMIT are applied on each node before each node's results are returned to the coordinator. Then the clauses are applied to the cumulative result set to get the final result set, which is returned to the calling application.

**Example:**

```
// This query works because p.status is part of projection list
select distinct p.ID, p.status from /region p where p.ID > 5 order by
p.status
// This query works providing status is part of the value indicated by *
select distinct * from /region where ID > 5 order by status
```

**Querying a Partitioned Region on a Single Node**

To direct a query to specific partitioned region node, you can execute the query within a function. Use the following steps:

1. Implement a function which executes a query using RegionFunctionContext.

```
/***
 * This function executes a query using its RegionFunctionContext
 * which provides a filter on data which should be queried.
 */
public class MyFunction extends FunctionAdapter {

    private final String id;

    @Override
    public void execute(FunctionContext context) {

        Cache cache = CacheFactory.getAnyInstance();
        QueryService queryService = cache.getQueryService();

        String qstr = (String) context.getArguments();

        try {
            Query query = queryService.newQuery(qstr);

            //If function is executed on region, context is
RegionFunctionContext
            RegionFunctionContext rContext = (RegionFunctionContext)context;

            SelectResults results = (SelectResults) query.execute(rContext)

            //Send the results to function caller node.
            context.getResultSender().sendResult((ArrayList)
(results).asList());
            context.getResultSender().lastResult(null);

        } catch (Exception e) {
            throw new FunctionException(e);
        }
    }

    @Override
    public boolean hasResult() {
        return true;
    }

    @Override
    public boolean isHA() {
        return false;
    }
}
```

```

public MyFunction(String id) {
    super();
    this.id = id;
}

@Override
public String getId() {
    return this.id;
}
}

```

- Decide on the data you want to query. Based on this decision, you can use `PartitionResolver` to configure the organization of buckets to be queried in the Partitioned Region.

For example, let's say that you have defined the `PortfolioKey` class:

```

public class PortfolioKey implements DataSerializable {
    private int id;
    private long startValidTime;
    private long endValidTime
    private long writtenTime

    public int getId() {
        return this.id;
    }
    ...
}

```

You could use the `MyPartitionResolver` to store all keys with the same ID in the same bucket. This `PartitionResolver` has to be configured at the time of Partition Region creation either declaratively using xml OR using APIs. See [Configuring Partitioned Regions](#) on page 215 for more information.

```

/** This resolver returns the value of the ID field in the key. With this
resolver,
* all Portfolios using the same ID are co-located in the same bucket.
*/
public class MyPartitionResolver implements PartitionResolver, Declarable {

    public Serializable getRoutingObject(EntryOperation operation) {
        return operation.getKey().getId();
    }
}

```

- Execute the function on a client or any other node by setting the filter in the function call.

```

/**
 * Execute MyFunction for query on specified keys.
 */
public class TestFunctionQuery {

    public static void main(String[] args) {

        ResultCollector rcollector = null;
        PortfolioKey portfolioKey1 = ...;

        //Filter data based on portfolioKey1 which is the key used in
        //region.put(portfolioKey1, portfolio1);
        Set filter = Collections.singleton(portfolioKey1);
    }
}

```

```
//Query to get all positions for portfolio ID = 1
String qStr = "SELECT positions FROM /myPartitionRegion WHERE ID =
1";
try {
    Function func = new MyFunction("testFunction");

    Region region =
CacheFactory.getAnyInstance().getRegion("myPartitionRegion");

    //Function will be routed to one node containing the bucket
    //for ID=1 and query will execute on that bucket.
    rcollector = FunctionService
        .onRegion(region)
        .withArgs(qStr)
        .withFilter(filter)
        .execute(func);

    Object result = rcollector.getResult();

    //Results from one or multiple nodes.
    ArrayList resultList = (ArrayList)result;

    List queryResults = new ArrayList();

    if (resultList.size()!=0) {
        for (Object obj: resultList) {
            if (obj != null) {
                queryResults.addAll((ArrayList)obj);
            }
        }
    }
    printResults(queryResults);
} catch (FunctionException ex) {
    getLogger().info(ex);
}
}
```

## Optimizing Queries on Data Partitioned by a Key or Field Value

You can improve query performance on data that is partitioned by key or a field value by creating a key index and then executing the query using the FunctionService with the key or field value used as filter.

The following is an example how to optimize a query that will be run on data partitioned by region key value. In the following example, data is partitioned by the "orderId" field.

1. Create a key index on the orderId field. See [Creating Key Indexes](#) on page 360 for more details.
  2. Execute the query using the function service with orderId provided as the filter to the function context. For example:

```
/**  
 * Execute MyFunction for query on data partitioned by orderId key  
 */  
public class TestFunctionQuery {  
  
    public static void main(String[] args) {  
  
        Set filter = new HashSet();
```

```

ResultCollector rcollector = null;

//Filter data based on orderId  = '12345'
filter.add(12345);

//Query to get all orders that match ID 12345 and amount > 1000
String qStr = "SELECT * FROM /Orders WHERE orderId = '12345' AND
amount > 1000";

try {
    Function func = new MyFunction("testFunction");

    Region region =
CacheFactory.getAnyInstance().getRegion("myPartitionRegion");

    //Function will be routed to one node containing the bucket
    //for ID=1 and query will execute on that bucket.
    rcollector = FunctionService
        .onRegion(region)
        .withArgs(qStr)
        .withFilter(filter)
        .execute(func);

    Object result = rcollector.getResult();

    //Results from one or multiple nodes.
    ArrayList resultList = (ArrayList)result;

    List queryResults = new ArrayList();

    if (resultList.size()!=0) {
        for (Object obj: resultList) {
            if (obj != null) {
                queryResults.addAll((ArrayList)obj);
            }
        }
    }
    printResults(queryResults);
}

} catch (FunctionException ex) {
    getLogger().info(ex);
}
}

```

## Performing an Equi-Join Query on Partitioned Regions

In order to perform equi-join operations on partitioned regions or partitioned regions and replicated regions, you need to use the `query.execute` method and supply it with a function execution context. You need to use GemFire's FunctionService executor because join operations are not yet directly supported for partitioned regions without providing a function execution context.

See [Partitioned Region Query Restrictions](#) on page 367 for more information on partitioned region query limitations.

For example, let's say your equi-join query is the following:

```
SELECT DISTINCT * FROM /QueryRegion1 r1,  
/QueryRegion2 r2 WHERE r1.ID = r2.ID
```

In this example QueryRegion2 is co-located with QueryRegion1, and both regions have same type of data objects.

On the server side:

```
Function prQueryFunction1 = new QueryFunction();
FunctionService.registerFunction(prQueryFunction1);

public class QueryFunction extends FunctionAdapter {
    @Override
    public void execute(FunctionContext context) {
        Cache cache = CacheFactory.getAnyInstance();
        QueryService queryService = cache.getQueryService();
        ArrayList allQueryResults = new ArrayList();
        ArrayList arguments = (ArrayList)(context.getArguments());
        String qstr = (String)arguments.get(0);
        try {
            Query query = queryService.newQuery(qstr);
            SelectResults result = (SelectResults)query
                .execute((RegionFunctionContext)context);
            ArrayList arrayResult = (ArrayList)result.asList();
            context.getResultSender().sendResult((ArrayList)result.asList());

            context.getResultSender().lastResult(null);
        } catch (Exception e) {
            // handle exception
        }
    }
}
```

On the server side, `Query.execute()` operates on the local data of the partitioned region.

On the client side:

```
Function function = new QueryFunction();
String queryString = "SELECT DISTINCT * FROM /QueryRegion1 r1,
    /QueryRegion2 r2 WHERE r1.ID = r2.ID";
ArrayList argList = new ArrayList();
argList.add(queryString);
Object result = FunctionService.onRegion(CacheFactory.getAnyInstance()
    .getRegion("QueryRegion1"))
    .withArgs(argList).execute(function).getResult();
ArrayList resultList = (ArrayList)result;
resultList.trimToSize();
List queryResults = null;
if (resultList.size() != 0) {
    queryResults = new ArrayList();
    for (Object obj : resultList) {
        if (obj != null) {
            queryResults.addAll((ArrayList)obj);
        }
    }
}
```

On the client side, note that you can specify a bucket filter while invoking `FunctionService.onRegion()`. In this case, the query engine relies on `FunctionService` to direct the query to specific nodes.

#### **Additional Notes on Using the `Query.execute` and `RegionFunctionContext` APIs**

You can also pass multiple parameters (besides the query itself) to the query function by specifying the parameters in the client-side code (`FunctionService.onRegion(...).withArgs()`). Then you can handle the parameters inside the function on the server side using `context.getArguments`. Note that it does not matter

which order you specify the parameters as long as you match the parameter handling order on the server with the order specified in the client.

## Query Debugging

You can debug a specific query at the query level by adding the <trace> keyword before the query string that you want to debug. Here is an example:

```
<trace> select * from /exampleRegion
```

You can also write:

```
<TRACE> select * from /exampleRegion
```

When the query is executed, GemFire will log a message in \$GEMFIRE\_DIR/system.log with the following information:

```
[info 2011/08/29 11:24:35.472 PDT CqServer <main> tid=0x1] Query Executed in 9.619656 ms; rowCount = 99; indexesUsed(0) "select * from /exampleRegion"
```

If you want to enable debugging for all queries, you can enable query execution logging by setting a System property on the command line during start-up:

```
gfsh>start server --name=server_name --J=-Dgemfire.Query.VERBOSE=true
```

Or you can set the property programmatically:

```
System.setProperty("gemfire.Query.VERBOSE", "true");
```

As an example, let us say you have an EmployeeRegion that that contains Employee objects as values and the objects have public fields in them like ID and status.

```
Employee.java
Class Employee {
    public int ID;
    public String status;
    - - - - -
}
```

In addition, you have created the following indexes for the region:

```
<index name="sampleIndex-1">
<functional from-clause="/test" expression="ID"/>
</index>
<index name="sampleIndex-2">
<functional from-clause="/test" expression="status"/>
</index>
```

After you have set gemfire.Query.VERBOSE to "true", you could see the following debug messages in the logs after running queries on the EmployeeRegion or its indexes:

- If indexes are not used in the query execution, you would see a debug message like this:

```
[info 2011/08/29 11:24:35.472 PDT CqServer <main> tid=0x1] Query Executed
in 9.619656 ms;
rowCount = 99; indexesUsed(0) "select * from /test k where ID > 0 and
status='active'"
```

- When single index is used in query execution, you might see a debug message like this:

```
[info 2011/08/29 11:24:35.472 PDT CqServer <main> tid=0x1] Query Executed
in 101.43499 ms;
rowCount = 199; indexesUsed(1):sampleIndex-1(Results: 199) "select count
* from /test k where ID > 0"
```

- When multiple indexes are used by a query, you might see a debug message like this:

```
[info 2011/08/29 11:24:35.472 PDT CqServer <main> tid=0x1] Query Executed  
in 79.43847 ms;  
rowCount = 199; indexesUsed(2):sampleIndex-2(Results:  
100),sampleIndex-1(Results: 199) "select * from /test k where ID > 0 OR  
status='active'"
```

In above log messages, the following information is provided:

- "rowCount" represents ResultSet size for the query.
- "indexesUsed(\n)" shows n indexes were used for finding the results of the query.
- Each index name and its corresponding results are reported respectively.
- The log can be identified with the original query string itself appended in the end.

# Chapter 36

## Continuous Querying

Continuous querying continuously returns events that match the queries you set up.

### How Continuous Querying Works

Clients subscribe to server-side events by using SQL-type query filtering. The server sends all events that modify the query results. CQ event delivery uses the client/server subscription framework.

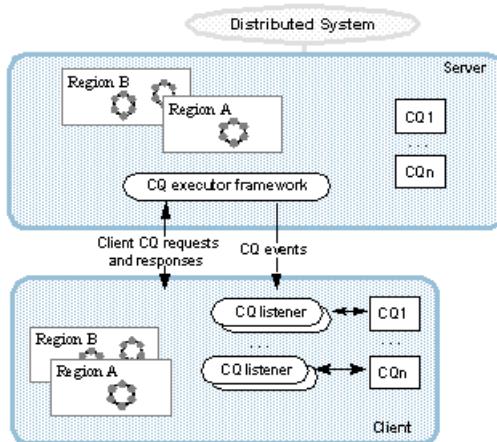
With CQ, the client sends a query to the server side for execution and receives the events that satisfy the criteria. For example, in a region storing stock market trade orders, you can retrieve all orders over a certain price by running a CQ with a query like this:

```
SELECT * FROM /tradeOrder t WHERE t.price > 100.00
```

When the CQ is running, the server sends the client all new events that affect the results of the query. On the client side, listeners programmed by you receive and process incoming events. For this example query on /tradeOrder, you might program a listener to push events to a GUI where higher-priced orders are displayed. CQ event delivery uses the client/server subscription framework.

### Logical Architecture of Continuous Querying

Your clients can execute any number of CQs, with each CQ assigned any number of listeners.



## Data Flow with CQs

CQs do not update the client region. This is in contrast to other server-to-client messaging like the updates sent to satisfy interest registration and responses to get requests from the client's Pool. CQs serve as notification tools for the CQ listeners, which can be programmed in any way your application requires.

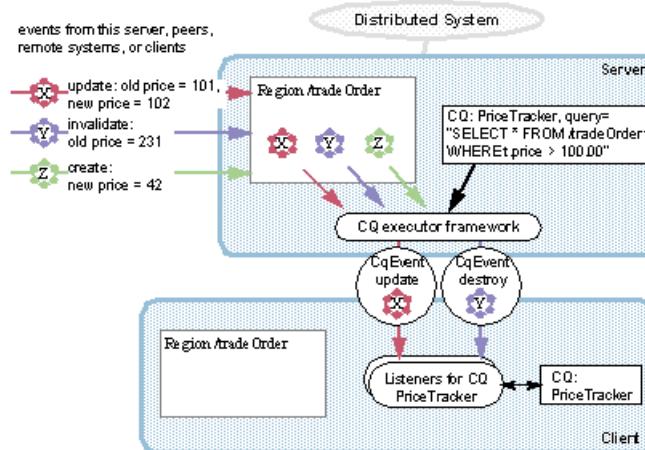
When a CQ is running against a server region, each entry event is evaluated against the CQ query by the thread that updates the server cache. If either the old or the new entry value satisfies the query, the thread puts a `CqEvent` in the client's queue. The `CqEvent` contains information from the original cache event plus information specific to the CQ's execution. Once received by the client, the `CqEvent` is passed to the `onEvent` method of all `CqListeners` defined for the CQ.

Here is the typical CQ data flow for entries updated in the server cache:

1. Entry events come to the server's cache from the server or its peers, distribution from remote sites, or updates from a client.
2. For each event, the server's CQ executor framework checks for a match with its running CQs.
3. If the old or new entry value satisfies a CQ query, a CQ event is sent to the CQ's listeners on the client side. Each listener for the CQ gets the event.

In the following figure:

- Both the new and old prices for entry X satisfy the CQ query, so that event is sent indicating an update to the query results.
- The old price for entry Y satisfied the query, so it was part of the query results. The invalidation of entry Y makes it not satisfy the query. Because of this, the event is sent indicating that it is destroyed in the query results.
- The price for the newly created entry Z does not satisfy the query, so no event is sent.



## CQ Events

CQ events do not change your client cache. They are provided as an event service only. This allows you to have any collection of CQs without storing large amounts of data in your regions. If you need to persist information from CQ events, program your listener to store the information where it makes the most sense for your application.

The `CqEvent` object contains this information:

- Entry key and new value.
- Base operation that triggered the cache event in the server. This is the standard `Operation` class instance used for cache events in GemFire.

- `CqQuery` object associated with this CQ event.
- `Throwable` object, returned only if an error occurred when the `CqQuery` ran for the cache event. This is non-null only for `CqListener` `onError` calls.
- Query operation associated with this CQ event. This operation describes the change affected to the query results by the cache event. Possible values are:
  - `CREATE`, which corresponds to the standard database
  - `INSERT` operation
  - `UPDATE`
  - `DESTROY`, which corresponds to the standard database `DELETE` operation

Region operations do not translate to specific query operations and query operations do not specifically describe region events. Instead, the query operation describes how the region event affects the query results.

<b>Query operations based on old and new entry values</b>	<b>New value does not satisfy the query</b>	<b>New value satisfies the query</b>
Old value does not satisfy the query	no event	<code>CREATE</code> query operation
Old value does satisfies the query	<code>DESTROY</code> query operation	<code>UPDATE</code> query operation

You can use the query operation to decide what to do with the `CqEvent` in your listeners. For example, a `CqListener` that displays query results on screen might stop displaying the entry, start displaying the entry, or update the entry display depending on the query operation.

## Implementing Continuous Querying

Use continuous querying in your clients to receive continuous updates to queries run on the servers. You can use a CQ in any of your client regions.

CQs are only run by a client on its servers. Before you begin, you should be familiar with [Querying](#) on page 325 and have your client/server system configured.

1. Configure the client pools you will use for CQs with `subscription-enabled` set to true.  
To have CQ and interest subscription events arrive as closely together as possible, use a single pool for everything. Different pools might use different servers, which can lead to greater differences in event delivery time.
2. Write your OQL query to retrieve the data you need from the server.  
The query must satisfy these CQ requirements in addition to the standard GemFire querying specifications:
  - The `FROM` clause must contain only a single region specification, with optional iterator variable.
  - The query must be a `SELECT` expression only, preceded by zero or more `IMPORT` statements. This means the query cannot be a statement like `"/tradeOrder.name"` or `"(SELECT * from /tradeOrder).size"`.
  - The CQ query cannot use:
    - Cross region joins
    - Drill-downs into nested collections
    - `DISTINCT`
    - Projections
    - Bind parameters

This is the basic syntax for the CQ query:

```
SELECT * FROM /fullRegionPath [iterator] [WHERE clause]
```

This example query could be used to get all trade orders where the price is over \$100:

```
SELECT * FROM /tradeOrder t WHERE t.price > 100.00
```

3. Write your CQ listeners to handle CQ events from the server.

Implement `com.gemstone.gemfire.cache.query.CqListener` in each event handler you need. In addition to your main CQ listeners, you might have listeners that you use for all CQs to track statistics or other general information.



**Note:** Be especially careful if you choose to update your cache from your `CqListener`. If your listener updates the region that is queried in its own CQ and that region has a Pool named, the update will be forwarded to the server. If the update on the server satisfies the same CQ, it may be returned to the same listener that did the update, which could put your application into an infinite loop. This same scenario could be played out with multiple regions and multiple CQs, if the listeners are programmed to update each other's regions.

This example outlines a `CqListener` that might be used to update a display screen with current data from the server. The listener gets the `queryOperation` and entry key and value from the `CqEvent` and then updates the screen according to the type of `queryOperation`.

```
// CqListener class
public class TradeEventListener implements CqListener
{
    public void onEvent(CqEvent cqEvent)
    {
        // com.gemstone.gemfire.cache Operation associated with the query op
        Operation queryOperation = cqEvent.getQueryOperation();
        // key and new value from the event
        Object key = cqEvent.getKey();
        TradeOrder tradeOrder = (TradeOrder)cqEvent.getNewValue();
        if (queryOperation.isUpdate())
        {
            // update data on the screen for the trade order . . .
        }
        else if (queryOperation.isCreate())
        {
            // add the trade order to the screen . . .
        }
        else if (queryOperation.isDestroy())
        {
            // remove the trade order from the screen . . .
        }
    }
    public void onError(CqEvent cqEvent)
    {
        // handle the error
    }
    // From CacheCallback public void close()
    {
        // close the output screen for the trades . . .
    }
}
```

When you install the listener and run the query, your listener will handle all of the CQ results.

4. If you need your CQs to detect whether they are connected to any of the servers that host its subscription queues, implement a `CqStatusListener` instead of a `CqListener`.

`CqStatusListener` extends the current `CqListener`, allowing a client to detect when a CQ is connected and/or disconnected from the server(s). The `onCqConnected()` method will be invoked when the CQ is connected, and when the CQ has been reconnected after being disconnected. The `onCqDisconnected()` method will be invoked when the CQ is no longer connected to any servers.

Taking the example from step 3, we can instead implement a `CqStatusListener`:

```
public class TradeEventListener implements CqStatusListener
{
    public void onEvent(CqEvent cqEvent)
    {
        // com.gemstone.gemfire.cache Operation associated with the query op
        Operation queryOperation = cqEvent.getQueryOperation();
        // key and new value from the event
        Object key = cqEvent.getKey();
        TradeOrder tradeOrder = (TradeOrder)cqEvent.getNewValue();
        if (queryOperation.isUpdate())
        {
            // update data on the screen for the trade order . . .
        }
        else if (queryOperation.isCreate())
        {
            // add the trade order to the screen . . .
        }
        else if (queryOperation.isDestroy())
        {
            // remove the trade order from the screen . . .
        }
    }
    public void onError(CqEvent cqEvent)
    {
        // handle the error
    }
    // From CacheCallback public void close()
    {
        // close the output screen for the trades . . .
    }

    public void onCqConnected() {
        //Display connected symbol
    }

    public void onCqDisconnected() {
        //Display disconnected symbol
    }
}
```

When you install the `CqStatusListener`, your listener will be able to detect its connection status to the servers that it is querying.

5. Program your client to run the CQ:
  - a) Create a `CqAttributesFactory` and use it to set your `CqListeners` and `CqStatusListener`.
  - b) Pass the attributes factory and the CQ query and its unique name to the `QueryService` to create a new `CqQuery`.
  - c) Start the query running by calling one of the execute methods on the `CqQuery` object.  
You can execute with or without an initial result set.
  - d) When you are done with the CQ, close it.

### Continuous Query Implementation

```

// Get cache and queryService - refs to local cache and
QueryService
// Create client /tradeOrder region configured to talk to the
server

// Create CqAttribute using CqAttributeFactory
CqAttributesFactory cqf = new CqAttributesFactory();

// Create a listener and add it to the CQ attributes callback
defined below
CqListener tradeEventListener = new TradeEventListener();
cqf.addCqListener(tradeEventListener);
CqAttributes cqa = cqf.create();
// Name of the CQ and its query
String cqName = "priceTracker";
String queryStr = "SELECT * FROM /tradeOrder t where t.price
> 100.00";

// Create the CqQuery
CqQuery priceTracker = queryService.newCq(cqName, queryStr,
cqa);

try
{ // Execute CQ, getting the optional initial result set
  // Without the initial result set, the call is
priceTracker.execute();
  SelectResults sResults =
priceTracker.executeWithInitialResults();
  List list1 = sResults.asList();
  for (int i=0; i < list1.size(); i++)
  {
    MyValueObject obj = (MyValueObject)list1.get(i);
    // ... do something with the entry key and value

  }
} catch (Exception ex)
{
  ex.printStackTrace();
}
// Now the CQ is running on the server, sending CqEvents to
the listener
. . .

// End of life for the CQ - clear up resources by closing
priceTracker.close();

```

With continuous queries, you can optionally implement:

- Highly available CQs by configuring your servers for high availability.
- Durable CQs by configuring your clients for durable messaging and indicating which CQs are durable at creation.

#### Related Topics

[com.gemstone.gemfire.cache.query](#)

[Continuous Querying \(GemFire Example\)](#)

## Managing Continuous Querying

This topic discusses CQ management options, CQ states, and retrieving initial result sets.

### States of a CQ

A CQ has three possible states, which are maintained on the server. You can check them from the client through `CqQuery.getState`.

Query State	What does this mean?	When does the CQ reach this state?	Notes
STOPPED	The CQ is in place and ready to run, but is not running.	When CQ is first created and after being stopped from a running state.	A stopped CQ uses system resources. Stopping a CQ only stops the CQ event messaging from server to client. All server-side CQ processing continues, but new CQ events are not placed into the server's client queue. Stopping a CQ does not change anything on the client side (but, of course, the client stops receiving events for the CQ that is stopped).
RUNNING	The CQ is running against server region events and the client listeners are waiting for CQ events.	When CQ is executed from a stopped state.	This is the only state in which events are sent to the client.
CLOSED	The CQ is not available for any further activities. You cannot rerun a closed CQ.	When CQ is closed by the client and when cache or connection conditions make it impossible to maintain or run.	The closed CQ does not use system resources.

### CQ Management Options

You manage your CQs from the client side. All calls are executed only for the calling client's CQs.

Task	For a single CQ use ...	For groups of CQs use ...
Create a CQ	<code>QueryService.newCq</code>	N/A
Execute a CQ	<code>CqQuery.execute</code> and <code>CqQuery.executeWithInitialResults</code>	<code>QueryService.executeCqs</code>
Stop a CQ	<code>CqQuery.stop</code>	<code>QueryService.stopCqs</code>
Close a CQ	<code>CqQuery.close</code>	<code>QueryService.closeCqs</code>
Access a CQ	<code>CqEvent.getCq</code> and <code>QueryService.getCq</code>	<code>QueryService.getCq</code>
Modify CQ Listeners	<code>CqQuery.getCqAttributesMutator</code>	N/A
Access CQ Runtime Statistics	<code>CqQuery.getStatistics</code>	<code>QueryService.getCqStatistics</code>
Get all durable CQs registered on the server	N/A	<code>QueryService.getAllDurableCqsFromServer</code>

### Managing CQs and Durable Clients Using gfsh

Using the `gfsh` command-line utility, you can perform the following actions:

- Close durable clients and durable client CQs. See [close](#) on page 675.

- List all durable CQs for a given durable client ID. See [list durable-cqs](#) on page 697.
- Show the subscription event queue size for a given durable client ID. See [show subscription-queue-size](#) on page 711.

### Retrieving an Initial Result Set of a CQ

You can optionally retrieve an initial result set when you execute your CQ. To do this, execute the CQ with the `executeWithInitialResults` method. The initial `SelectResults` returned is the same that you would get if you ran the query ad hoc, by calling `QueryService.newQuery.execute` on the server cache, but with the key included. This example retrieves keys and values from an initial result set:

```
SelectResults cqResults = cq.executeWithInitialResults();
for (Object o : cqResults.asList()) {
    Struct s = (Struct)o; // Struct with Key, value pair
    Portfolio p = (Portfolio)s.get("value"); // get value from the Struct
    String id = (String)s.get("key"); // get key from the Struct
}
```

If you are managing a data set from the CQ results, you can initialize the set by iterating over the result set and then updating it from your listeners as events arrive. For example, you might populate a new screen with initial results and then update the screen from a CQ listener.

If a CQ is executed using the `ExecuteWithInitialResults` method, the returned result may already include the changes with respect to the event. This can arise when updates are happening on the region while CQ registration is in progress. The CQ does not block any region operation as it could affect the performance of the region operation. Design your application to synchronize between the region operation and CQ registration to avoid duplicate events from being delivered.

# Chapter 37

---

## Transactions

With GemFire transactions, you execute transactions against the data in the distributed cache in the same way you would use transactions in a database. The vFabric GemFire API for distributed transactions has the familiar relational database methods, `begin`, `commit`, and `rollback`.

### About Transactions

This section covers the features of GemFire transactions.

GemFire transactions provide the following features:

- Basic transaction properties: atomicity, consistency, isolation, and durability
- Rollback and commit operations along with standard GemFire cache operations
- Ability to suspend and resume transactions
- High concurrency and high performance
- Transaction statistics gathering and archiving
- Compatibility with Java Transaction API (JTA) transactions, using either GemFire JTA or a third-party implementation.

### Types of Transactions

GemFire supports two kinds of transactions: GemFire cache transactions and JTA global transactions.

GemFire cache transactions are used to group the execution of cache operations and to gain control offered by transactional commit and rollback. Applications create cache transactions by using an instance of the GemFire `CacheTransactionManager` to start and terminate transactions. During the transaction, cache operations are performed and distributed through GemFire as usual. See [GemFire Cache Transactions](#) on page 384 for details on GemFire cache transactions and how these transactions work.

JTA global transactions allow you to use the standard Java interface (JTA) to coordinate GemFire transactions with JDBC transactions. When performing JTA global transactions, you have the option of using GemFire's own implementation of JTA or a third party's implementation (typically application servers like WebLogic or JBoss) of JTA. For global transactions, applications use `java:/UserTransaction` to start and terminate transactions while GemFire cache operations are performed in the same manner as regular GemFire cache transactions. See [JTA Global Transactions with GemFire](#) on page 401 for details on JTA Global transactions.



**Note:** You can also coordinate a GemFire cache transaction with an external database by specifying database operations in cache and transaction application plug-ins (CacheWriters/CacheListeners and TransactionWriters/TransactionListeners.) This is an alternative to using JTA transactions. See [How to Run a GemFire Cache Transaction that Coordinates with an External Database](#) on page 386.

## GemFire Cache Transactions

Use GemFire cache transactions to group the execution of cache operations and to gain the control offered by transactional commit and rollback.

GemFire cache transactions control operations within the GemFire cache while the GemFire distributed system handles data distribution in the usual way. Running these cache operations as a batch simplifies administration and improves system performance.

### Cache Transaction Performance

Cache transaction performance can vary depending on the type of regions you are using.

The most common region configurations for use with transactions are distributed replicated and partitioned:

- Replicated regions are better suited for running transactions on small to mid-size data sets. To ensure all or nothing behavior, at commit time, distributed transactions use the global reservation system of the GemFire distributed lock service. This works well as long as the data set is reasonably small.
- Partitioned regions are the right choice for highly-performant, scalable operations. Transactions on partitioned regions use only local locking, and only send messages to the redundant data stores at commit time. Because of this, these transactions perform much better than distributed transactions. There are no global locks, so partitioned transactions are extremely scalable as well.

### Data Location for Cache Transactions

The location where you can run your transaction depends on where you are storing your data.

Transactions must be run on a data set hosted entirely by one member.

- For replicated or other distributed regions, the transaction uses only the data set in the member where the transaction is run.
- For partitioned regions, you must co-locate all your transactional data in a single member.
- For transactions run on partitioned and distributed region mixes, you must co-locate the partitioned region data and make sure the distributed region data is available in any member hosting the partitioned region data.

For transactions involving partitioned regions, any member with the regions defined can run the transactional application, regardless of whether the member hosts data for the regions. If the transactional data resides on a remote member, the transaction is carried out by proxy in the member hosting the data. The member hosting the data is referred to as the transactional data host.

### How to Run a GemFire Cache Transaction

This topic describes how to run a GemFire cache transaction.

Applications manage transactions on a per-cache basis. A GemFire cache transaction starts with a `CacheTransactionManager.begin` method and continues with a series of operations, which are typically region operations such as region create, clear and destroy. The begin, commit, and rollback are directly controlled by the application. A commit, failed commit, or voluntary rollback by the transaction manager ends the transaction.

You can run transactions on any type of cache region except regions with **global** scope. A transaction attempted on these regions throws an `UnsupportedOperationException`.

Before you begin, have your members and regions defined where you want to run transactions.



**Note:** The discussions on transactions in this guide use replicated and partitioned regions. If you use non-replicated distributed regions, follow the guidelines for replicated regions.

1. **Configure the cache copy-on-read behavior in the members hosting the transactional data, or perform cache updates that avoid in-place changes.** This allows the transaction manager to control when your cache updates are visible outside the transaction.

2. **Configure your regions for transactions in the members hosting the transactional data.** Depending on the type of regions and features you are using, you should configure your regions differently for transactions.

If you use...

then you should...

**replicated regions**

Use distributed-ack scope. The region shortcuts specifying REPLICATE use distributed-ack scope. This is particularly important if you have more than one data producer. With one data producer, you can safely use distributed-no-ack.

**partitioned regions**

Custom partition and co-locate data between regions so all the data for any single transaction is hosted by a single data store. If you run the transaction from another member, it will run by proxy in the member hosting the data. The partitioned region must be defined for the application that runs the transaction, but the data can be hosted in a remote member.

**persistent regions**

Configure GemFire to allow transactions on persistent regions. By default, GemFire does not allow transactions on persistent regions. You can enable the use of transactions on persistent regions by setting the gemfire property gemfire.ALLOW\_PERSISTENT\_TRANSACTIONS to true.

**a mix of partitioned and replicated regions**

Make sure your replicated regions are hosted in every member that hosts the partitioned region data. All data for a single transaction must reside in a single host.

**delta propagation**

Set the region attribute cloning-enabled to true. This lets GemFire do conflict checks at commit. Without this, when you run the transaction, you'll get this exception: UnsupportedOperationInTransactionException("Delta without cloning cannot be used in transaction");

**global JTA transactions but only want to run a GemFire cache transaction**

Set the region attribute ignore-jta to true for all regions that you do *not* want to participate in JTA global transactions. It is false by default. For instructions on how to run a JTA global transaction, see How to Run a JTA Transaction.

3. **Update your cache event handler and transaction event handler implementations to handle your transactions.**

Cache event handlers are all used for transactions. Cache listeners are called after the commit, instead of after each cache operation, and they receive the conflated transaction events. Cache writers and loaders are called as usual, as the operations are done.

Follow these additional guidelines when writing your cache event handler callbacks:

- Make sure cache callbacks are transactionally aware. A transactional operation can launch callbacks that are not transactional.
- Make sure cache listeners will operate properly with the entry event conflation done for transactional operations. Two entry events for the same key are conflated by removing the existing event and adding the new event to the end of the list.

See [Using Cache Writer and Cache Listener Plug-Ins](#) on page 389 for more information.

If desired, you can also configure transaction event handlers. Transaction event handlers are cache-wide. You can install one transaction writer and any number of transaction listeners. Use the following guidelines:

- Program with synchronization for thread-safety. Your listener and writer methods may be called simultaneously by different threads for different transactions.
- Keep your transactional callback implementations lightweight and avoid doing anything that might cause them to block.

See [Configuring Transaction Plug-In Event Handlers](#) on page 390 for more information.

4. **Using the CacheTransactionManager, write your transaction.** For example:

```
CacheTransactionManager txManager =
    custRegion.getCache().getCacheTransactionManager();

try {
    txManager.begin();
    // ... do work
```

```

        txManager.commit();
    } catch (CommitConflictException conflict)

```

Follow these guidelines when writing your transaction:

- Start each transaction with a begin operation.
- Run the GemFire function and other operations that you want included within the transaction.
- Consider whether you will want to suspend and resume the transaction. If some operations should not be part of the transaction, you may want to suspend the transaction while performing non-transactional operations. After the non-transactional operations are complete, you can resume the transaction. See [Basic Suspend and Resume Transaction Example](#) on page 411 for an example.
- If your transaction runs on a mix of partitioned and replicated regions, perform your first transaction operation on a partitioned region. This sets the host for the entire transaction.
- If you did not configure copy-on-read to true, be sure your cache updates avoid in-place changes.
- Take into account the behavior of transactional and non-transactional operations. All transactional operations that are run after the begin and before the commit or rollback are included in the transaction.
- End each transaction with a commit or a rollback. Do not leave any transaction in an uncommitted and unrolled back state. Transactions do not time out, so will remain in the system for the life of your application.

## 5. Review all of your code for compatibility with transactions.

When you commit a transaction, while the commit is taking place, the changes are visible in the distributed cache. This provides better performance than locking everything involved with the transaction updates, but it means that another process accessing data used in the transaction might get some data in the pre-transaction state and some in the post-transaction state.

For example, suppose key 1 and 2 are written to in a transaction so both of their values change from A to B. In another thread, it is possible to read key 1 with value B and key 2 with value A, while the transaction is being committed. This is possible due to the nature of GemFire reads. This choice sacrifices atomic visibility in favor of performance; reads do not block writes, and writes do not block reads.

## How to Run a GemFire Cache Transaction that Coordinates with an External Database

This topic describes how to coordinate a GemFire cache transaction with an external database by using CacheWriter/CacheListener and TransactionWriter/TransactionListener plug-ins. This provides an alternative to using JTA transactions.

There are a few things you should be careful about while working with GemFire cache transactions and external databases:

- When you set up the JDBC connection, make sure that auto-commit is disabled. For example, in Java:

```

Connection getConnection() throws SQLException {
    Connection con = ... // create the connection
    con.setAutoCommit(false);
    return con;
}

```

- The BEGIN statement, database operations and the PREPARE statement must all happen in the same connection session. In order to accomplish this, you will need to obtain the same JDBC connection session across multiple CacheWriter and TransactionWriter/TransactionListener invocations. One way to do this would be to lookup the connection (from a user managed Map) based on `cacheTransactionManager.getTransactionId()`.
- Make sure that the prepare transaction feature is enabled in your external database. For example, it is disabled in PostgreSQL by default. In PostgreSQL, the following property must be modified to enable it:

```

max_prepared_transactions = 1 # 1 or more enables, zero (default) disables
this feature.

```

Use the following procedure to write a GemFire cache transaction that coordinates with an external database:

1. Configure GemFire regions as necessary as described in [How to Run a GemFire Cache Transaction](#) on page 384.
2. Begin the transaction.

a) In GemFire:

```
CacheTransactionManager txManager =
    custRegion.getCache().getCacheTransactionManager();

try {
    txManager.begin();
    // ... do GemFire operations
```

- b) If you have not previously committed a previous transaction in this connection, start a database transaction by issuing a BEGIN statement.
3. Perform GemFire cache operations; each cache operation invokes the CacheWriter. Program your CacheWriter to do the corresponding database operations.
4. Commit the transaction:

```
txManager.commit();
} catch (CommitConflictException conflict)
```

At this point, the TransactionWriter is invoked. The TransactionWriter returns a TransactionEvent, which contains all the operations in the transaction. Call PREPARE TRANSACTION within your TransactionWriter code.

5. After a transaction is successfully committed in GemFire, TransactionListener is invoked. Call COMMIT PREPARED in the TransactionListener to commit the database transaction.

## Working with GemFire Cache Transactions

This section contains guidelines and additional information on working with GemFire and its cache transactions.

### Setting Global Copy on Read

Copy-on-read is not specifically a transaction setting, but if you are using transactions, you probably want copy-on-read behavior. To enable global copy-on-read for all reads, modify cache.xml or use the Java API.

Using cache.xml:

```
<cache lock-lease="120" lock-timeout="60"
search-timeout="300" copy-on-read="true">
```

API:

```
Cache c = CacheFactory.getInstance(system) c.setCopyOnRead(true);
```

The copy-on-read attribute and the operations affected by the attribute setting are discussed in detail in [Managing Data Entries](#) on page 157.

### Making a Safe Change Within a Transaction Using CopyHelper.copy

If you do not have copy-on-read set globally and are using replicated regions, you must explicitly make copies of the cache objects that you are modifying in your transactions. You can use CopyHelper.copy to make these copies.

```
CacheTransactionManager cTxMgr = cache.getCacheTransactionManager();
cTxMgr.begin(); Object o = (StringBuffer)r.get("stringBuf");
StringBuffer s = (StringBuffer) CopyHelper.copy(o);
s.append("Changes unseen before commit. Read Committed.");
r.put("stringBuf", s); cTxMgr.commit();
```

## Transactions and Functions

You can run a function inside a transaction and you can run a transaction inside a function, as long as your combination of functions and transactions does not result in nested transactions. You cannot call a function from inside a transaction if the function runs its own transaction.

You can also have multiple functions participate within a single transaction.

If you are suspending and resuming a transaction with multiple function calls, all function calls participating in the transaction must execute on the same member.

See [Transactional Using a Function Example](#) on page 414 for an example.

## Using Queries and Indexes with Transactions

Queries and indexes reflect the cache and ignore the changes made by ongoing transactions. If you do a query from inside a transaction, the query does not reflect the changes you made inside that transaction.

## Collections and Region.Entry Instances in Transactions

Collections and region entries used in a transaction must be created inside the transaction. After the transaction has completed, the application can no longer use any region entry or collection or associated iterator created within the transaction. If you try to use these, you get an `IllegalStateException`.

Region collection operations include `Region.keySet`, `Region.entrySet`, and `Region.values`. You can create instances of `Region.Entry` through the `Region.getEntry` operation or by looking at the contents of the result returned by a `Region.entrySet` operation.

## Using Eviction and Expiration Operations

Entry expiration and LRU eviction affect the committed state. They are not part of the transaction, and they cannot be rolled back.

### About Eviction

LRU eviction operations do not cause write conflicts with existing transactions, despite destroying or invalidating entries. LRU eviction is deferred on entries modified by the transaction until the commit completes. Because anything touched by the transaction has had its LRU clock reset, eviction of those entries is not likely to happen immediately after the commit.

When a transaction commits its changes in a region with distributed scope, the operation can invoke eviction controllers in the remote caches, as well as in the local cache.

### Configure Expiration

Local expiration actions do not cause write conflicts, but distributed expiration can cause conflicts and prevent transactions from committing in the members receiving the distributed operation.

- Using local expiration with transactions. When you are using transactions on a region, make expiration local, if possible. For every instance of that region, configure an expiration action of local invalidate or local destroy. In a `cache.xml` declaration, use a line similar to this:

```
<expiration-attributes timeout="60" action="local-invalidate" />
```

- In regions modified by a transaction, local expiration is suspended. Expiration operations are batched and deferred per region until the transactions complete. Once cleanup starts, the manager processes pending expirations. Transaction that need to change the region wait until the expirations are complete.
- Using distributed expiration with transactions. With partitioned and replicated regions, you cannot use local expiration. When you are using distributed expiration, the expiration is not suspended during a transaction, and expiration operations distributed from another member can cause write conflicts. In replicated regions, you can avoid conflicts by setting up your distributed system this way:
  - Choose an instance of the region to drive region-wide expiration. Use a replicated region, if there is one.

- Configure distributed expiration only in that region instance. The expiration action must be either invalidate or destroy. In a cache.xml declaration, use a line similar to this:

```
<expiration-attributes timeout="300" action="destroy"/>
```

- Run your transactions in the member where you configured expiration.

## Transactions and Consistent Regions

A transaction that modifies a region having consistency checking enabled generates all necessary version information for region updates when the transaction commits.

If a transaction modifies a normal, preloaded or empty region, the transaction is first delegated to a GemFire member that holds a replicate for the region. This behavior is similar to the transactional behavior for partitioned regions, where the partitioned region transaction is forwarded to a member that hosts the primary for the partitioned region update.

The limitation for transactions with a normal, preloaded or empty region is that, when consistency checking is enabled, a transaction cannot perform a `localDestroy` or `localInvalidate` operation against the region. GemFire throws an `UnsupportedOperationException` exception in such cases. An application should use a `Destroy` or `Invalidate` operation in place of a `localDestroy` or `localInvalidate` when consistency checks are enabled.

## Suspending and Resuming Transactions

vFabric GemFire APIs provide the ability to suspend and resume transactions. The ability to suspend and resume is useful when a thread must perform some operations that should not be part of the transaction before the transaction can complete.

When a transaction is suspended, it loses the transactional view of the cache. None of the previous operations (before calling `suspend`) are visible to the thread. Subsequently any operations that are performed by the thread do not participate in the suspended transaction.

When a transaction is resumed, the resuming thread assumes the transactional view. A transaction that is suspended on a member must be resumed on the same member. Before resuming a transaction, you may want to check if the transaction exists on the member and whether it is suspended.

You may optionally use the `tryResume` method.

If the member with the primary copy of the data crashes, the transactional view that applied to that data is lost. The secondary member for the data will not be able to resume any transactions suspended on the crashed member. You will need to take remedial steps to retry the transaction on a new primary copy of the data.

If a suspended transaction is not touched for a period of time, GemFire cleans it up automatically. By default, the timeout for a suspended transaction is 30 minutes and can be configured using the system property `gemfire.suspendedtxTimeout`. For example, to modify the default, you could set `gemfire.suspendedtxTimeout=60` (time in minutes).

## Using Cache Writer and Cache Listener Plug-Ins

All standard vFabric GemFire application plug-ins work with transactions. In addition, the transaction interface offers specialized plug-ins that support transactional operation.



**Note:** No direct interaction exists between client transactions and client application plug-ins. When a client runs a transaction, GemFire calls the plug-ins that are installed on the transaction's server delegate and its server host. Client application plug-ins are not called for operations inside the transaction or for the transaction as a whole. When the transaction is committed, the changes to the server cache are sent to the client cache according to client interest registration. These events can result in calls to the client's `CacheListeners`, as with any other events received from the server.

The `EntryEvent` that the callbacks receive has a unique GemFire transaction ID, so the cache listener can associate each event, as it occurs, with a particular transaction. `EntryEvents` that are not part of a transaction return null instead of a transaction ID.

- `CacheLoader`. When a cache loader is called by a transaction operation, values loaded by the cache loader may cause a write conflict when the transaction commits.
- `CacheWriter`. During a transaction, if a cache writer exists, its methods are invoked as usual for all operations, as the operations are called in the transactions. The `netWrite` operation is not used. The only cache writer used is the one in the member where the transactional data resides.
- `CacheListener`. The cache listener callbacks - local and remote - are triggered after the transaction commits. The system sends the conflated transaction events, in the order they were stored.

### Configuring Transaction Plug-In Event Handlers

GemFire has two types of transaction plug-ins-- Transaction Writers and Transaction Listeners. You can optionally install one transaction writer and one or more transaction listener per cache.

Like JTA global transactions, you can use transaction plug-in event handlers to coordinate GemFire cache transaction activity with an external data store. However, you typically use JTA global transactions when GemFire is running as a peer data store with your external data stores. Transaction writers and listeners are typically used when GemFire is acting as a front end cache to your backend database.



**Note:** You can also use transaction plug-in event handlers when running JTA global transactions.

### TransactionWriter

When you commit a transaction, if a transaction writer is installed in the cache where the data updates were performed, it is called. The writer can do whatever work you need, including aborting the transaction.

The transaction writer is the last place that an application can rollback a transaction. If the transaction writer throws any exception, the transaction is rolled back. For example, you might use a transaction writer to update a backend data source before the GemFire cache transaction completes the commit. If the backend data source update fails, the transaction writer implementation can throw a `TransactionWriterException` to veto the transaction.

A typical usage scenario would be to use the transaction writer to prepare the commit on the external database. Then in a transaction listener, you can apply the commit on the database.

### Transaction Listeners

When the transaction ends, its thread calls the transaction listener to perform the appropriate follow-up for successful commits, failed commits, or voluntary rollbacks. The transaction that caused the listener to be called no longer exists by the time the listener code executes.

Transaction listeners have access to the transactional view and thus are not affected by non-transactional update operations. `TransactionListener` methods cannot make transactional changes or cause a rollback. They can, however, start a new transaction. Multiple transactions on the same cache can cause concurrent invocation of `TransactionListener` methods, so implement methods that do the appropriate synchronization of the multiple threads for thread-safe operation.

A transaction listener can preserve the result of a transaction, perhaps to compare with other transactions, or for reference in case of a failed commit. When a commit fails and the transaction ends, the application cannot just retry the transaction, but must build up the data again. For most applications, the most efficient action is just to start a new transaction and go back through the application logic again.

The rollback and failed commit operations are local to the member where the transactional operations are run. When a successful commit writes to a distributed or partitioned region, however, the transaction results are distributed to other members the same as other updates. The transaction listener on the receiving members reflect

the changes the transaction makes in that member, not the originating member. Any exceptions thrown by the transaction listener are caught by GemFire and logged.

### How Transaction Events Are Managed

Transactional cache operations are handled somewhat differently inside transactions than out.

- During the Transaction. While the transaction is running, each transactional operation is passed to the cache writer local to the transactional view, if one is available. As with cache operations outside of transactions, the cache writer can abort the operation. Each operation the cache writer allows is applied to the transactional view in the cache and appended to the CacheEvent list in the TransactionEvent object.

- Event Conflation. The cache events are conflated, so if a key already has an event in the list, that event is removed and the current operation is added to the end of the list. So this series of calls inside a transaction:

```
Region.create (A, W);
Region.put (A, valX);
Region.put (B, valQ);
Region.invalidate (A);
Region.put (A, valY);
```

results in these events stored in the CacheEvent list, with only the most recent operation left:

```
put (B, valQ)
put (A, valY)
```

- At commit and after commit. When the transaction is committed, GemFire passes the TransactionEvent to the transaction writer local to the transactional view, if one is available. After commit, GemFire:

- Passes the TransactionEvent to each installed transaction listener.
- Walks the CacheEvent list, calling all locally installed listeners for each operation listed.
- Distributes the TransactionEvent to all interested caches.



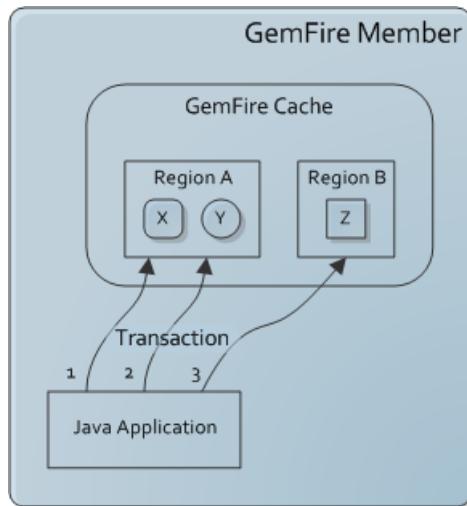
**Note:** For GemFire and global JTA transactions, the EntryEvents contain the GemFire transaction ID. JTA transaction events do not contain the JTA transaction ID.

### How GemFire Cache Transactions Work

This section provides an explanation on how transactions work on GemFire caches.

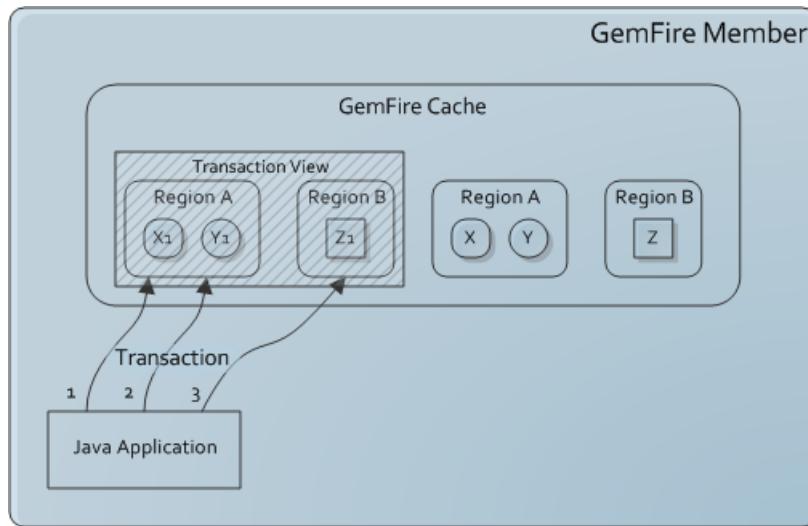
vFabric GemFire transactions operate on data in local member caches. All the regions in a member cache can participate in a transaction. A Java application can operate on the cache using multiple transactions. A transaction is associated with only one thread, and a thread can operate on only one transaction at a time. Child threads do not inherit existing transactions.

The diagram shows a transaction operating against a local member cache.



### Transaction View

A transaction is isolated from changes made concurrently to the cache. Each transaction has its own private view of the cache, including the entries it has read and the changes it has made. The first time the transaction touches an entry in the cache, either to read or write, it produces a snapshot of that entry's state in the transaction's view. The transaction remembers the entry's original state and uses it at commit time to discover write conflicts. The transaction maintains its current view of the entry, which reflects only the changes made within the transaction.



### Committing Transactions

When a commit succeeds, the changes recorded in the transaction view are merged into the cache. If the commit fails or the transaction is rolled back, all of its changes are dropped.

When a transaction is committed, the transaction management system uses a two-phase commit protocol:

1. Reserves all the entries involved in the transaction from changes by any other transactional thread. For distributed regions, it reserves the entries in the entire distributed system. For partitioned regions, it reserves them on the data store, where the transaction is running.

2. Checks the cache for conflicts on affected keys, to make sure all entries are still in the same state they were in when this transaction first accessed them.
3. If any conflict is detected, the manager rolls back the transaction.
4. If no conflict is detected, the manager:
  - a. Calls the `TransactionWriter` in the member where the transaction is running. This allows the system to write through transactional updates to an external data source.
  - b. Updates the local cache and distributes the updates to the other members holding the data. Cache listeners are called for these updates, in each cache where the changes are made, the same as for non-transactional operations.
  - c. Calls the `TransactionListeners` in the member where the transaction is running.
5. Releases the transaction reservations on the entries.

The manager updates the local cache and distributes the updates to other members in a non-atomic way.

- If other threads read the keys the transaction is modifying, they may see some in their pre-transaction state and some in their post-transaction state.
- If other, non-transactional, sources update the keys the transaction is modifying, the changes may intermingle with this transaction's changes. The other sources can include distributions from remote members, loading activities, and other direct cache modification calls from the same member. When this happens, after your commit finishes, the cache state may not be what you expected.

If the transaction fails to complete any of the steps, a `CommitConflictException` is thrown to the calling application.

Once the members involved in the transaction have been asked to commit, the transaction completes even if one of the participating members were to leave the system during the commit. The transaction completes successfully so long as all remaining members are in agreement.

Each member participating in the transaction maintains a membership listener on the transaction coordinator. If the transaction coordinator goes away after issuing the final commit call, the transaction completes in the remaining members.

### **Transactions by Region Type**

A transaction is managed on a per-cache basis, so multiple regions in the cache can participate in a single transaction. The data scope of a vFabric GemFire cache transaction is the cache that hosts the transactional data. For partitioned regions, this may be a remote host to the host running the transaction application. Any transaction that includes one or more partitioned regions is run on the member storing the primary copy of the partitioned region data. Otherwise, the transaction host is the same one running the application.

- The member running the transaction code is called the transaction initiator.
- The member that hosts the data—and the transaction—is called the transactional data host.

So the transactional data host may be local or remote to the transaction initiator. In either case, when the transaction commits, data distribution is done from the transactional data host in the same way.



**Note:** If you have consistency checking enabled in your region, the transaction will generate all necessary version information for the region update when the transaction commits. See [Transactions and Consistent Regions](#) on page 389 for more details.

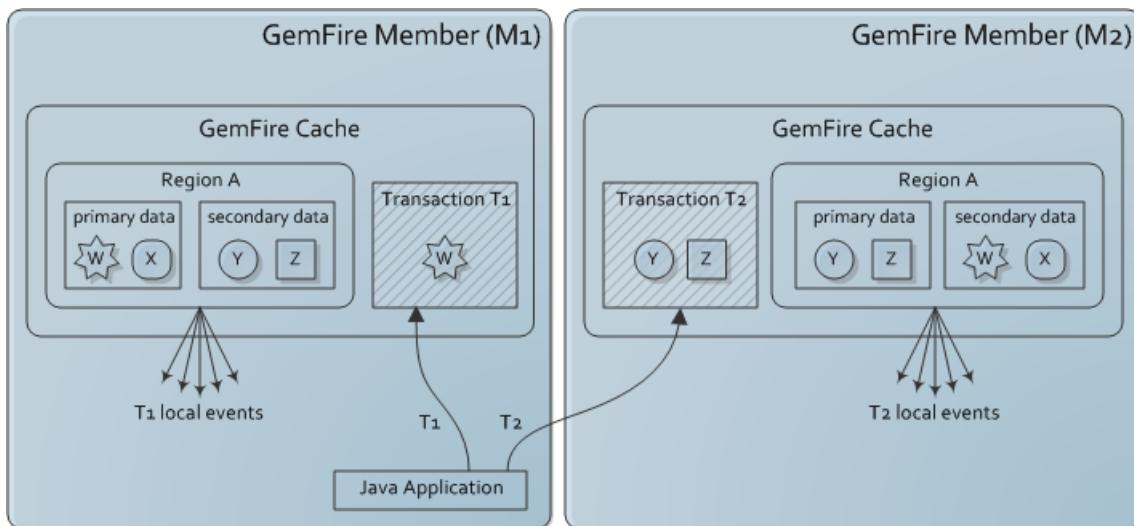
### *Transactions and Partitioned Regions*

In partitioned regions, transaction operations are done first on the primary data store then distributed to other members from there, regardless of which member initializes the cache operation. This is the same as is done for normal cache operations on partitioned regions.

In this figure, M1 runs two transactions.

- The first, T1, works on data whose primary buckets are stored in M1, so M1 is both initiator and data host for the transaction.
- The second transaction, T2, works on data whose primary buckets are stored in M2, so M1 is the transaction initiator and M2 is the transactional data host.

*Transaction on a Partitioned Region:*



The transaction is managed on the data host. This includes the transactional view, all operations, and all local cache event handling. In the figure, when T2 is committed, the cache on M2 is updated and the transaction events distributed throughout the system, exactly as if the transaction had originated on M2.

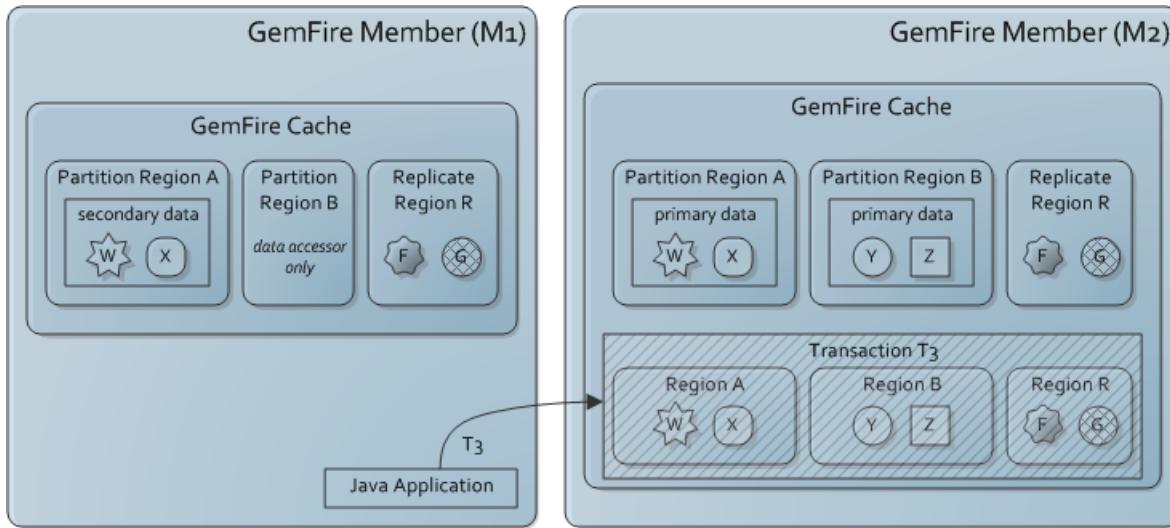
The first region operation in the transaction determines the transactional data host. All other operations must also work with that as their transactional data host:

- All partitioned region data managed inside the transaction must use the transactional data host as their primary data store. In the figure above, if transaction T2 tried to put entry W or transaction T1 tried to put entry Z, they would get a `TransactionDataNotColocatedException`. For information on partitioning your data so it is grouped properly for your transactions, see [Understanding Custom Partitioning and Data Colocation](#) on page 218. In addition, the data must not be moved during the transaction. Plan any partitioned region rebalancing to avoid rebalancing while transactions are running. See [Rebalancing Partitioned Region Data](#) on page 230.
- All non-partitioned region data managed inside the transaction must be available on the transactional data host and must be distributed. Operations on regions with local scope are not allowed in transactions with partitioned regions.

The next figure shows a transaction that uses two partitioned regions and one replicated region. As with the single region example, all local event handling is done on the transactional data host.

For a transaction in these data keys to work, the first operation must be on one of the partitioned regions, to establish M2 as the transactional data host. Running the first operation on a key in the replicated region would establish M1 as the transactional data host, and subsequent operations on the partitioned region data would fail with a `TransactionDataNotColocated` exception.

*Transaction on a Partitioned Region with Other Regions:*



### Transactions and Replicated Regions

For replicated regions, the transaction and its operations are applied to the local member and the resulting transaction state is distributed to other members according to the attributes of each region.



**Note:** If possible, use `distributed-ack` scope for your regions where you will run transactions. The `REPLICATE` region shortcuts use `distributed-ack` scope.

The region's scope affects how data is distributed during the commit phase. Transactions are supported for these region scopes:

- **`distributed-ack`** . Handles transactional conflicts both locally and between members. The `distributed-ack` scope is designed to protect data consistency. This scope provides the highest level of coordination among transactions in different members. When the commit call returns for a transaction run on all distributed-ack regions, you can be sure that the transaction's changes have already been sent and processed. In addition, any callbacks in the remote member have been invoked.
- **`distributed-no-ack`** . Handles transactional conflicts locally, less coordination between members. This provides the fastest transactions with distributed regions, but doesn't work for all situations. This scope is appropriate for:
  - Applications with only one writer
  - Applications with multiple writers that write to different data sets
- **`local`** . No distribution, handles transactional conflicts locally. Transactions on regions with local scope have no distribution, but they perform conflict checks in the local member. You can have conflict between two threads when their transactions change the same entry, like object Y in this figure.

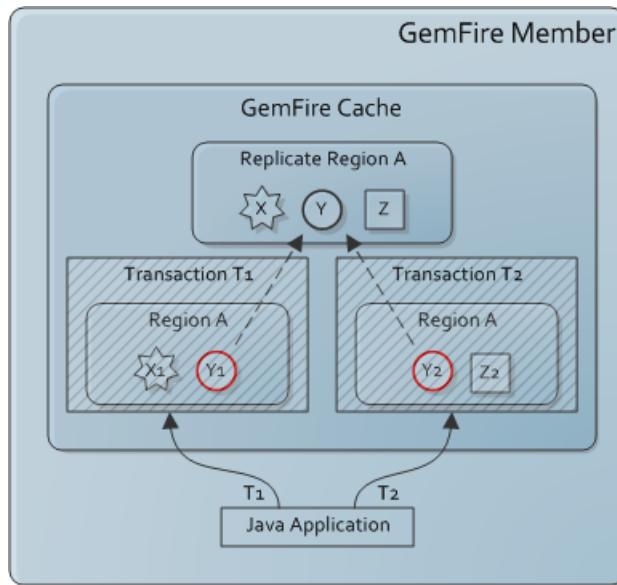
Transactions on non-replicated regions (regions that use the old API with DataPolicy EMPTY, NORMAL and PRELOADED) are always transaction initiators, and the transaction data host is always a member with a replicated region. This is similar to the way transactions using the PARTITION\_PROXY shortcut are forwarded to members with primary bucket.



**Note:** When you have transactions operating on EMPTY, NORMAL or PARTITION regions, make sure that the GemFire property `conserve-sockets` is set to false to avoid distributed deadlocks. An empty region is a region created with the API `RegionShortcut.REPLICATE_PROXY` or a region with that uses the old API of `DataPolicy` set to EMPTY.

## Conflicting Transactions with Local Scope

When encountering conflicts with local scope, the first transaction to start the commit process "wins." The other transaction's commit fails with a conflict, and its changes are dropped. In the diagram below, the resulting value for entry "Y" depends on which transaction commits first.



### *Transactions and Persistent Regions*

By default, GemFire does not allow transactions on persistent regions. You can enable the use of transactions on persistent regions by setting the gemfire property `gemfire.ALLOW_PERSISTENT_TRANSACTIONS` to true.

When executing transactions on persistent regions, we recommend using the TransactionWriter to log all transactions along with a time stamp. This will allow you to recover in the event that all nodes fail simultaneously while a transaction is being committed. You can use the log to recover the data manually.

### **Client Transactions**

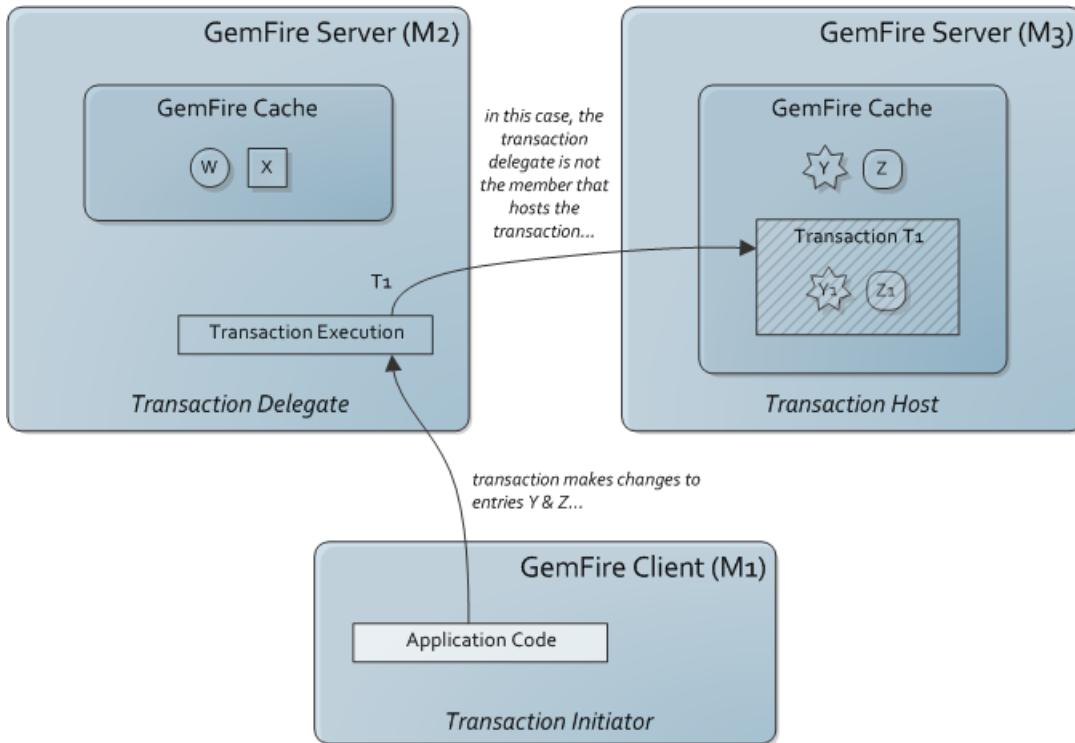
The syntax for writing client transactions is the same on the Java client as with any other GemFire member, but the underlying behavior in a client-run transaction is different from general transaction behavior.

For general information about running a transaction, refer to [How to Run a GemFire Cache Transaction](#) on page 384.

#### *How GemFire Runs Client Transactions*

When a client performs a transaction, the transaction is delegated to a server that acts as the transaction initiator in the server system. As with regular, non-client transactions, this server delegate may or may not be the transaction host.

In this figure, the application code on the client (M1) makes changes to data entries Y and Z within a transaction. The delegate performing the transaction (M2) does not host the primary copy of the data being modified. The transaction takes place on the server containing this data (M3).



#### *Client Cache Access During a Transaction*

To maintain cache consistency, GemFire blocks access to the local client cache during a transaction. The local client cache may reflect information inconsistent with the transaction in progress. When the transaction completes, the local cache is accessible again.

#### *Client Transactions and Client Application Plug-Ins*

Any plug-ins installed in the client are not invoked by the client-run transaction. The client that initiates the transaction receives changes from its server based on transaction operations the same as any other client - through mechanisms like subscriptions and continuous query results. The client transaction is performed by the server delegate, where application plug-ins operate the same as if the server were the sole initiator of the transaction.

#### *Client Transaction Failures*

In addition to the failure conditions common to all transactions, client transactions can fail if the transaction delegate fails. If the delegate performing the transaction fails, the transaction code throws a transaction exception. See [Transaction Exceptions](#) on page 410.

#### **Comparing Transactional and Non-Transactional Operations**

Between the begin operation and the commit or rollback operation are a series of ordinary GemFire operations. When they are launched from a transaction, the GemFire operations can be classified into two types:

- Transactional operations that affect the transactional view
- Non-transactional operations that do not affect the transactional view

An operation that acts directly on the cache does not usually act on the transactional view.

#### *Transactional Operations*

The `CacheTransactionManager` methods are the only ones used specifically for cache transactions. Otherwise, you use the same GemFire methods as usual. Most methods that run within a transaction affect the transactional view and do not change the cache until the transaction commits. Methods that behave this way are

considered transactional operations. Transactional operations are classified in two ways: whether they modify the transactional view or the cache itself, and whether they create write conflicts with other transactions.

In general, methods that create, destroy, invalidate, update, or read region entries are transactional.

Transactional operations that can cause write conflicts are those that modify an entry, such as put, a load done to satisfy a get operation, create, delete, local delete, invalidate and local invalidate.

Transactional read operations do not cause conflicts directly, but they can modify the transactional view. Read operations look for the entry in the transaction view first and then, if necessary, go to the cache. If the entry is returned by a cache read, it is stored as part of the transactional view. At commit time, the transaction uses the initial snapshot of the entry in the view to discover write conflicts.

### *Non-Transactional Operations*

A few methods, when running in the context of a transaction, have no effect on the transactional view but have an immediate effect on the cache. They are considered non-transactional operations. Often, non-transactional operations are administrative, such as `Region.destroy` and `Region.invalidate`. These operations are not supported within a transaction. If you call them, the system throws an exception of this type:

`UnsupportedOperationExceptionInTransactionException(destroyRegion() is not supported while in a transaction.)`

## Entry Operations



**Note:** Transactional entry operations can be rolled back.

Operations	Methods	Transactional	Write Conflict
create	<code>Region.create</code> , <code>put</code> , <code>putAll</code> , <code>Map.put</code> , <code>putAll</code>	yes	yes
modify	<code>Region.put</code> , <code>putAll</code> , <code>Map.put</code> , <code>putAll</code> , <code>Region.Entry.setValue</code> , <code>Map.Entry.setValue</code>	yes	yes
load	<code>Region.get</code> , <code>Map.get</code>	yes	yes
creation or update using <code>netSearch</code>	<code>Region.get</code> , <code>Map.get</code>	yes	no
destroy: local and distributed	<code>Region.localDestroy</code> , <code>destroy</code> , <code>remove</code> , <code>Map.remove</code>	yes	yes
invalidate: local and distributed	<code>Region.localInvalidate</code> , <code>invalidate</code>	yes	yes
set user attribute	<code>Region.Entry.setUserAttribute</code>	yes	yes
read of a single entry	<code>Region.get</code> , <code>getEntry</code> , <code>containsKey</code> , <code>containsValue</code> , <code>containsValueForKey</code>	yes	no
read of a collection of entries	<code>Region.keySet</code> , <code>entrySet</code> , <code>values</code>	Becomes transactional when you access the keys or values within the collection.	no

Some transactional write operations also do a read before they write, and these can complete a transactional read even when the write fails. The following table of entry operations notes the conditions under which this can happen.



**Note:** These operations can add a snapshot of an entry to the transaction's view even when the write operation does not succeed.

Operations	Methods	Reads Without Writing
create	Region.create	when it throws an EntryExistsException
destroy: local and distributed	Region.localDestroy, destroy	when it throws an EntryNotFoundException
invalidate: local and distributed	Region.localInvalidate, invalidate	when it throws an EntryNotFoundException or the entry is already invalid

## Region Operations

When you create a region in a transaction, any data from the getInitialImage operation goes directly into the cache, rather than waiting for the transaction to commit.

Operations	Methods	Affected	Write Conflict
destroy: local and distributed	Region.localDestroyRegion, destroyRegion	cache	yes
invalidate: local and distributed	Region.localInvalidateRegion, invalidateRegion	cache	yes
clear: local and distributed	Region.localClear, clear, Map.clear	cache and transaction	no
close	Region.close	cache	yes
mutate attribute	Region.getAttributesMutator methods	cache	no
set user attribute	Region.setUserAttribute	cache	no

## Cache Operations

When you create a region in a transaction, any data from the getInitialImage operation goes directly into the cache, rather than waiting for the transaction to commit.

Operations	Methods	Affected State	Write Conflict
create	createRegion	committed	no
close	close	committed	yes

## No-Ops

Any operation that has no effect in a non-transactional context remains a no-op in a transactional context. For example, if you do two localInvalidate operations in a row on the same region, the second localInvalidate is a no-op. No-op operations do not:

- Cause a listener invocation
- Cause a distribution message to be sent to other members
- Cause a change to an entry
- Cause any conflict

A no-op can do a transactional read.

## GemFire Cache Transaction Semantics

vFabric GemFire transaction semantics differs in some ways from the semantics of traditional relational databases.

### Atomicity

A common database technique to achieve atomicity is two-phase locking on rows. This provides coordination between the current transaction and other transactions with overlapping changes. Holding write locks prevents reads of transactional changes until all changes are complete, providing all or nothing behavior. If problems occur during a transaction, perhaps due to relational constraints or storage limitations, the locks protect other processes from reading until the problems can be resolved. The resolution often involves reverting data back to the original pre-transaction state. Rolling back a database transaction in this case would require releasing write locks.

A common distributed technique for all or nothing behavior is the Two Phase Commit Protocol, which requires all processes participating in the transaction to agree to commit or abort the transaction.

The all or nothing behavior of GemFire transactions is implemented using a per thread transaction state which collects all transactional changes. The set of Entry changes are atomically reserved by a reservation system. The reservation blocks other, intersecting transactions from proceeding, allowing the commit to check for conflicts and reserve resources in an all-or-nothing fashion prior to making changes to the data.

Since GemFire is distributed, the transaction state is distributed to the appropriate members and processed in a batch, to protect against application failure in the middle of the commit. After all changes have been made, locally and remotely, the reservation is released.

Rolling back the transaction is as simple as discarding the transaction state.

Reads do not use the reservation system and so are very fast.

This system has traits similar to the Multiversion Concurrency Control algorithm:

- Each thread can write new versions of entry data without impacting read performance
- Reads view only the most recent committed version of an entry

### Consistency

Database consistency is often described in terms of referential integrity. References between tables are often described in terms of primary and foreign key constraints. A database transaction that updates multiple columns, potentially in different tables, must ensure that all updates are completed to maintain consistency in an atomic fashion.

For GemFire the application controls the consistency between regions and their entries. GemFire transactions use the reservation system to ensure that changes to all entries in all regions are committed. Once a reservation is obtained, GemFire does a conflict check to ensure that the entries scheduled for change are in the same state as they were when the transaction first accessed them.

### Isolation

GemFire isolates transactions at the process thread level. So while a transaction is in progress, its changes are only visible inside the thread that is running the transaction. Threads inside the same process and in other processes do not see the changes until after the commit finishes. Relational database transactions often isolate changes per JDBC connection.

GemFire reads, like get and getEntry, are volatile, meaning they do not use locks. This makes reads very fast with respect to other entry operations. It also allows transactional writes to proceed without being slowed down by in-progress reads.

Visibility, related to isolation, is the atomic nature in which committed writes are viewable by read operations. Reads during a GemFire transaction have repeatable read isolation, so once the committed value reference is

read for a given key, it always returns the same reference. If a transaction write, like put or invalidate, deletes a value for a key that has already been read, subsequent reads return the transactional reference.

## Durability

Relational databases provide durability by using disk storage for recovery and transaction logging. GemFire does not support on-disk or in-memory durability for transactions.

## JTA Global Transactions with GemFire

Use JTA global transactions to coordinate GemFire cache transactions and JDBC transactions.

JTA is a standard Java interface you can use to coordinate GemFire cache transactions and JDBC transactions globally under one umbrella. JTA provides direct coordination between the GemFire cache and another transactional resource, such as a database. The parties involved in a JTA transaction include:

- The Java application, responsible for starting the global transaction
- The JTA transaction manager, responsible for opening, committing, and rolling back transactions
- The transaction resource managers, including the GemFire cache transaction manager and the JDBC resource manager, responsible for managing operations in the cache and database respectively

Using JTA, your application controls all transactions in the same standard way, whether the transactions act on the GemFire cache, a JDBC resource, or both together. When a JTA global transaction is done, the GemFire transaction and the database transaction are both complete.

When using JTA global transactions with GemFire, you have two options:

1. Coordinate with an external JTA transaction manager in a container (such as WebLogic or JBoss)
2. Have GemFire act as the JTA transaction manager

An application creates a global transaction by using `javax.transaction.UserTransaction` bound to the JNDI context `java:/UserTransaction` to start and terminate transactions. During the transaction, cache operations are done through GemFire as usual as described in [GemFire Cache Transactions](#) on page 384.



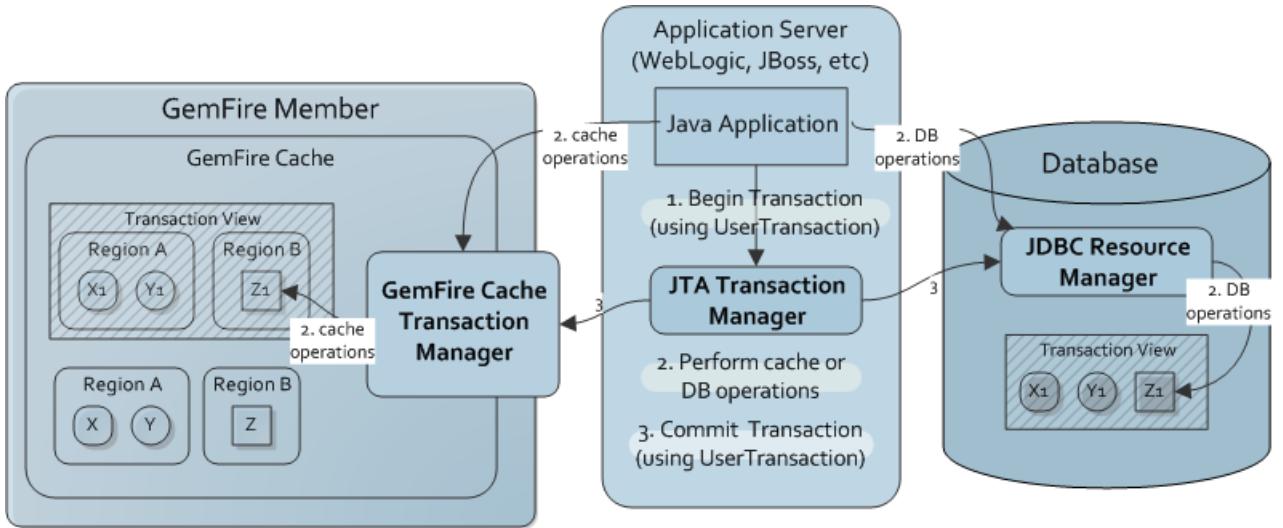
**Note:** See the Sun documentation for more information on topics such as JTA, `javax.transaction`, committing and rolling back global transactions, and the related exceptions.

## Coordinating with External JTA Transactions Managers

GemFire can work with the JTA transaction managers of several containers like JBoss, WebLogic, GlassFish, and so on.

At startup GemFire looks for a `TransactionManager` (`javax.transaction.TransactionManager`) that has been bound to its JNDI context. When GemFire finds such an external transaction manager, all GemFire region operations (such as get and put) will participate in global transactions hosted by this external JTA transaction manager.

This figure shows the high-level operation of a JTA global transaction whose resources include a GemFire cache and a database.



An externally coordinated JTA global transaction is run in the following manner:

1. Each region operation looks up for presence of a global transaction. If one is detected, then a GemFire transaction is started automatically, and we register a `javax.transaction.Synchronization` callback with the external JTA transaction manager.
2. At transaction commit, GemFire gets a `beforeCommit()` callback from the external JTA transaction manager. GemFire does all locking and conflict detection at this time. If this fails, an exception is thrown back to JTA transaction manager, which then aborts the transaction.
3. After a successful `beforeCommit()` callback, JTA transaction manager asks other data sources to commit their transaction.
4. GemFire then gets a `afterCommit()` callback in which changes are applied to the cache and distributed to other members.

You can disable JTA in any region that should not participate in JTA transactions. See [Turning Off JTA Transactions](#) on page 409.

### How to Run a JTA Transaction Coordinated by an External Transaction Manager

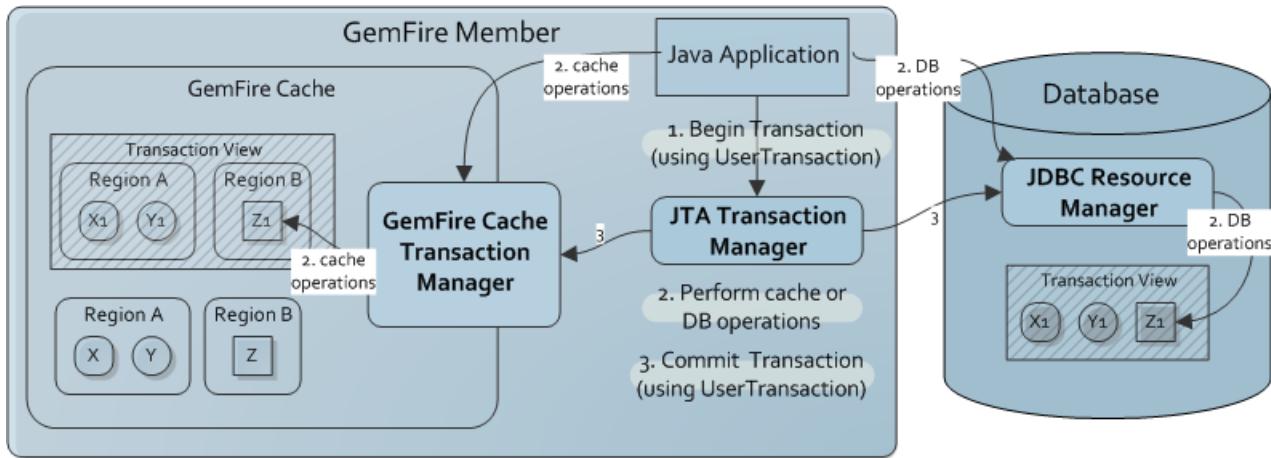
Use the following procedure to run a GemFire global JTA transaction coordinated by an external JTA transaction manager.

1. Configure the external data sources in the external container. Do not configure the data sources in `cache.xml`. They are not guaranteed to get bound to the JNDI tree.
2. Configure GemFire for any necessary transactional behavior in `cache.xml`. For example, enable `copy-on-read` and specify a transaction listener, if you need one. See [Setting Global Copy on Read](#) on page 387 and [Configuring Transaction Plug-In Event Handlers](#) on page 390 for details.
3. Make sure that JTA transactions are not disabled for the regions that will participate in the transaction. See [Turning Off JTA Transactions](#) on page 409 for details.
4. Start the transaction through the external container.
5. Initialize the GemFire cache. GemFire will automatically join the transaction.
6. Execute operations in the cache and the database as usual.
7. Commit the transaction through the external container.

### Using GemFire as the JTA Transaction Manager

You can also use GemFire as the JTA transaction manager.

GemFire ships with its own implementation of a JTA transaction manager. However, note that this implementation is not XA compliant; therefore, it does not persist any state, which could lead to an inconsistent state after recovering a crashed member.



The GemFire JTA transaction manager is initialized when the GemFire cache is initialized. Until then, JTA is not available for use. The application starts a JTA transaction by using the `UserTransaction.begin` method. The `UserTransaction` object is the application's handle to instruct the JTA transaction manager on what to do.

The GemFire JTA implementation also supports the J2EE Connector Architecture (J2CA) `ManagedConnectionFactory`.

The GemFire Enterprise implementation of JTA has the following limitations:

- Only one JDBC database instance per transaction is allowed, although you can have multiple connections to that database.
- Multiple threads cannot participate in a transaction.
- Transaction recovery after a crash is not supported.

In addition, JTA transactions are subject to the limitations of GemFire cache transactions such as not being supported on regions with global scope. When a global transaction needs to access the GemFire cache, JTA silently starts a GemFire cache transaction.

## How to Run a JTA Global Transaction Using GemFire as the JTA Transaction Manager

This topic describes how to run a JTA global transaction in GemFire.

The GemFire JTA is initialized when the GemFire cache is initialized. Until then, that JTA is not available for use. The application starts a JTA transaction through `UserTransaction.begin`. `UserTransaction` is the application's handle to tell the JTA transaction manager what to do.

To run a global transaction, perform the following steps:

1. Configure the external data sources in `cache.xml`. See [Configuring Database Connections Using JNDI](#) on page 404 for examples.
2. Include the JAR file for any data sources in your CLASSPATH .
3. Configure GemFire for any necessary transactional behavior. Enable `copy-on-read` for your cache and specify a transaction listener, if you need one. See [Setting Global Copy on Read](#) on page 387 and [Configuring Transaction Plug-In Event Handlers](#) on page 390 for details.
4. Make sure that JTA transactions are not disabled in the `cache.xml` file or the application code.
5. Initialize the GemFire cache.

- Get an initial context through `com.gemstone.cache.Cache.getJNDIContext`. For example:

```
Context ctx = cache.getJNDIContext();
```

This returns `javax.naming.Context` and gives you the JNDI associated with the cache. The context contains the `TransactionManager`, `UserTransaction`, and any configured JDBC resource manager.

- Look up the `UserTransaction` context:

```
UserTransaction txManager =
(UserTransaction)ctx.lookup("java:/UserTransaction");
```

With `UserTransaction`, you can begin, commit, and rollback transactions.

If a global transaction exists, when you use the cache, it automatically joins the transaction. Operations on a region automatically detect and become associated with the existing global transaction through JTA synchronization. If the global transaction has been marked for rollback, however, the GemFire cache is not allowed to enlist with that transaction. Any cache operation that causes an attempt to enlist throws a `FailedSynchronizationException`.

The GemFire cache transaction's commit or rollback is triggered when the global transaction commits or rolls back. When the global transaction is committed using the `UserTransaction` interface, the transactions of any registered JTA resources are committed, including the GemFire cache transaction. If the cache or database transaction fails to commit, `UserTransaction` throws a `TransactionRolledBackException`. If a commit or rollback is attempted directly on a GemFire transaction that is registered with JTA, that action throws an `IllegalStateException`.

See [GemFire JTA Transaction Example](#) on page 415.

### *Configuring Database Connections Using JNDI*

When using JTA transactions, you can configure database JNDI data sources in `cache.xml`. The `DataSource` object points to either a JDBC connection or, more commonly, a JDBC connection pool. The connection pool is usually preferred, because a program can use and reuse a connection as long as necessary and then free it for another thread to use.

The following are a list of `Datasource` connection types.

- **XAPooledDataSource**. Pooled SQL connections.
- **ManagedDataSource**. JNDI binding type for the J2EE Connector Architecture (J2CA). `ManagedConnectionFactory`. For information on the `ManagedConnection` interface, see: <http://docs.oracle.com/javaee/6/api/javax/resource/spi/ManagedConnection.html>.
- **PooledDataSource**. Pooled SQL connections.
- **SimpleDataSource**. Single SQL connection. No pooling of SQL connections is done. Connections are generated on the fly and cannot be reused.

The `jndi-name` attribute of the `jndi-binding` element is the key binding parameter. If the value of `jndi-name` is a `DataSource`, it is bound as `java:/myDatabase`, where `myDatabase` is the name you assign to your data source. If the data source cannot be bound to JNDI at runtime, GemFire logs a warning. For information on the `DataSource` interface, see:

<http://docs.oracle.com/javase/6/docs/api/javax/sql/DataSource.html>

GemFire supports JDBC 2.0 and 3.0.



**Note:** Include any data source jar files in your CLASSPATH.

### *Example DataSource Configurations in cache.xml*

The following sections show example `cache.xml` files configured for each of the `DataSource` connection types.

### JNDI-Binding Properties Configuration Example

You specify the JNDI binding properties through the config-property tag, as shown in this example. To find what other properties you need to set, see the vendor documentation for your database. You can add as many config-property tags as required.

```
<jndi-binding . . .
    <config-property>

<config-property-name>databaseName</config-property-name>
<config-property-type>java.lang.String</config-property-type>

    <config-property-value>newDB</config-property-value>
</config-property>
</jndi-binding>
```

### XAPooledDataSource Connection Example

The example shows a cache.xml file configured for a pool of XAPooledDataSource connections connected to the data resource newDB. The log-in and blocking timeouts are set lower than the defaults. The connection information, including user-name and password, is set in the cache.xml file, instead of waiting until connection time. The password is encrypted; for details, see [Encrypting Passwords for Use in cache.xml](#) on page 505.

```
<?xml version="1.0"?>
<!DOCTYPE cache PUBLIC
"-//GemStone Systems, Inc.//GemFire Declarative Caching
7.0//EN"
"http://www.gemstone.com/dtd/cache7_0.dtd">
<cache lock-lease="120" lock-timeout="60"
search-timeout="300">
    <region name="root">
        <region-attributes scope="distributed-no-ack"
data-policy="cached" initial-capacity="16"
load-factor="0.75" concurrency-level="16"
statistics-enabled="true">
            .
            .
        </region>
        <jndi-bindings>
            <jndi-binding type="XAPooledDataSource"
jndi-name="newDB2trans" init-pool-size="20"
max-pool-size="100" idle-timeout-seconds="20"
blocking-timeout-seconds="5"
login-timeout-seconds="10"
xa-datasource-class="org.apache.derby.jdbc.EmbeddedXADataSource"
user-name="mitul"
password="encrypted(83f0069202c571faf1ae6c42b4ad46030e4e31c17409e19a)">
                <config-property>

<config-property-name>description</config-property-name>
<config-property-type>java.lang.String</config-property-type>
```

```

<config-property-value>pooled_transact</config-property-value>
    </config-property>
    <config-property>
        <config-property-name>databaseName</config-property-name>
        <config-property-type>java.lang.String</config-property-type>
            <config-property-value>newDB</config-property-value>
        </config-property>
        <config-property>
            <config-property-name>vendor_specific_property1</config-property-name>
            <config-property-type>type</config-property-type>
            <config-property-value>value</config-property-value>
        </config-property>
        .
        .
        </jndi-binding>
    </jndi-bindings>
</cache>
```

### ManagedDataSource Connection Example

ManagedDataSource connections for the J2CA ManagedConnectionFactory are configured as shown in the example. This configuration is similar to XAPooledDataSource connections, except the type is ManagedDataSource, and you specify a managed-conn-factory-class instead of an xa-datasource-class.

```

<?xml version="1.0"?>
<!DOCTYPE cache PUBLIC "-//GemStone Systems, Inc./GemFire
Declarative Caching 7.0//EN"
"http://www.gemstone.com/dtd/cache7_0.dtd">
<cache lock-lease="120" lock-timeout="60"
search-timeout="300">
    <region name="root">
        <region-attributes scope="distributed-no-ack"
data-policy="cached" initial-capacity="16"
load-factor="0.75" concurrency-level="16"
statistics-enabled="true">
        .
        .
        </region>
        <jndi-bindings>
            <jndi-binding type="ManagedDataSource"
jndi-name="DB3managed"
init-pool-size="20" max-pool-size="100"
idle-timeout-seconds="20"
blocking-timeout-seconds="5" login-timeout-seconds="10"
managed-conn-factory-class="com.myvendor.connection.ConnFactory"
user-name="mitul"
password="encrypted(83f0069202c571faf1ae6c42b4ad46030e4e31c17409e19a)">
            <config-property>
```

```

<config-property-name>description</config-property-name>
<config-property-type>java.lang.String</config-property-type>

<config-property-value>pooled_transact</config-property-value>
    </config-property>
    <config-property>

<config-property-name>databaseName</config-property-name>
<config-property-type>java.lang.String</config-property-type>

<config-property-value>newDB</config-property-value>
    </config-property>
    <config-property>

<config-property-name>vendor_specific_property1</config-property-name>

<config-property-type>type</config-property-type>
<config-property-value>value</config-property-value>
    </config-property>
    .
    .
    </jndi-binding>
</jndi-bindings>
</cache>

```

### PooledDataSource and SimpleDataSource Example

Use the PooledDataSource and SimpleDataSource connections for operations executed outside of any transaction. This example shows a cache.xml file configured for a pool of PooledDataSource connections to the data resource newDB. For this non-transactional connection pool, the log-in and blocking timeouts are set higher than for the transactional connection pools in the two previous examples. The connection information, including user-name and password, is set in the cache.xml file, instead of waiting until connection time. The password is encrypted; for details, see [Encrypting Passwords for Use in cache.xml](#) on page 505.

```

<?xml version="1.0"?>
<!DOCTYPE cache PUBLIC "-//GemStone Systems, Inc.//GemFire Declarative Caching 7.0//EN"
"http://www.gemstone.com/dtd/cache7_0.dtd">
<cache lock-lease="120" lock-timeout="60"
search-timeout="300">
    <region name="root">
        <region-attributes scope="distributed-no-ack"
data-policy="cached"
initial-capacity="16" load-factor="0.75"
concurrency-level="16" statistics-enabled="true">
        .
        </region>
    <jndi-bindings>
        <jndi-binding type="PooledDataSource">

```

```

jndi-name="newDB1" init-pool-size="2"
max-pool-size="7" idle-timeout-seconds="20"
blocking-timeout-seconds="20"
login-timeout-seconds="30"
conn-pooled-datasource-class="org.apache.derby.jdbc.EmbeddedConnectionPoolDataSource"
user-name="mitul"
password="encrypted(83f0069202c571faf1ae6c42b4ad46030e4e31c17409e19a)">

    <config-property>

<config-property-name>description</config-property-name>
<config-property-type>java.lang.String</config-property-type>

<config-property-value>pooled_nontransact</config-property-value>
    </config-property>
    <config-property>

<config-property-name>databaseName</config-property-name>
<config-property-type>java.lang.String</config-property-type>

<config-property-value>newDB</config-property-value>
    </config-property>
    <config-property>

<config-property-name>vendor_specific_property1</config-property-name>

<config-property-type>value</config-property-type>
<config-property-value>value</config-property-value>
    </config-property>
    .
    .
    </jndi-binding>
</jndi-bindings>
</cache>

```

### SimpleDataSource Connection Example

The example below shows a very basic configuration in the `cache.xml` file for a SimpleDataSource connection to the data resource `oldDB`. You only need to configure a few properties like a `jndi-name` for this connection pool, `oldDB1`, and the `databaseName`, `oldDB`. This password is in clear text.

A simple data source connection does not generally require vendor-specific property settings. If you need them, add `config-property` tags as shown in the earlier examples.

```

<?xml version="1.0"?>
<!DOCTYPE cache PUBLIC "-//GemStone Systems, Inc.//GemFire
Declarative Caching 7.0//EN"
"http://www.gemstone.com/dtd/cache7_0.dtd">
<cache lock-lease="120" lock-timeout="60"
search-timeout="300">
    <region name="root">

```

```

        <region-attributes scope="distributed-no-ack"
data-policy="cached" initial-capacity="16"
load-factor="0.75" concurrency-level="16"
statistics-enabled="true">
    . .
</region-attributes>
</region>
<jndi-bindings>
    <jndi-binding type="SimpleDataSource"
jndi-name="oldDB1"
jdbc-driver-class="org.apache.derby.jdbc.EmbeddedDriver"
user-name="mitul"
password="password"
connection-url="jdbc:derby:newDB;create=true">
    . .
</jndi-binding>
</jndi-bindings>
</cache>
```

### Behavior of GemFire Cache Writers and Loaders Under JTA

When GemFire participates in a global transaction, you can still have GemFire cache writers and cache loaders operating in the usual way.

For example, in addition to the transactional connection to the database, the region could also have a cache writer and cache loader configured to exchange data with that same database. As long as the data source is transactional, which means that it can detect the transaction manager, the cache writer and cache loader participate in the transaction. If the JTA rolls back its transaction, the changes made by the cache loader and the cache writer are rolled back.

If you are using a GemFire cache or transaction listener with global transactions, be aware that the `EntryEvent` returned by a transaction has the GemFire transaction ID, not the JTA transaction ID.

### Turning Off JTA Transactions

You can configure regions to not participate in any JTA global transaction.

The `ignore-jta` region attribute is a boolean that tells the cache to ignore any in-progress JTA transactions when performing cache operations. It is primarily used for cache loaders, cache writers, and cache listeners that need to perform non-transactional operations on a region, such as caching a result set. It is set per region, so some regions can participate in JTA transactions, while others avoid participating in them. This example sets the `ignore-jta` region attribute in the `cache.xml` file.

`cache.xml` :

```

<region name="bridge_region">
    <region-attributes scope="local" ignore-jta="true"
statistics-enabled="true"/>
        <cache-writer> . . . </cache-writer>
    </region-attributes>
</region>
```

API:

Using the API, you can turn off JTA transactions using `RegionFactory` and its method `setIgnoreJTA(boolean)`. The current setting for a region can be fetched from a region's `RegionAttributes` by using the `getIgnoreJTA` method.

## Monitoring and Troubleshooting Transactions

This topic covers errors that may occur when running transactions in GemFire.

Unlike database transactions, GemFire does not write a transaction log to disk. To get the full details about committed operations, use a transaction listener to monitor the transaction events and their contained cache events for each of your transactions.

### Statistics on Cache Transactions

During the operation of GemFire cache transactions, if statistics are enabled, transaction-related statistics are calculated and accessible from the CachePerfStats statistic resource. Because the transaction's data scope is the cache, these statistics are collected on a per-cache basis.

### Commit

In a failed commit, the exception lists the first conflict that caused the failure. Other conflicts can exist, but are not reported.

### Capacity Limits

A transaction can create data beyond the capacity limit set in the region's eviction attributes. The capacity limit does not take effect until commit time. Then, any required eviction action takes place as part of the commit.

### Interaction with the Resource Manager

The GemFire resource manager, which controls overall heap use, either allows all transactional operations or blocks the entire transaction. If a cache reaches the critical threshold in the middle of a commit, the commit is allowed to finish before the manager starts blocking operations.

### Transaction Exceptions

The following sections list possible transaction exceptions.

#### Exceptions Indicating Transaction Failure

- **TransactionDataNodeHasDepartedException**. This exception means the transactional data host has departed unexpectedly. Clients and members that run transactions but are not the transactional data hosts can get this exception. You can avoid this by working to ensure your transactional data hosts are stable and remain running when transactions are in progress.
- **TransactionDataNotColocatedException**. You will get this error if you try to run a transaction on data that is not all located in the same member. Partition your data so it is all located in a single member. See [Transactions and Partitioned Regions](#) and [Understanding Custom Partitioning and Data Colocation](#) on page 218.
- **TransactionDataRebalancedException**. You get this error if your transactional data is moved to another member for rebalancing during the transaction. Manage your partitioned region data to avoid rebalancing during a transaction. See [Rebalancing Partitioned Region Data](#) on page 230.

#### Exceptions Indicating Unknown Transaction Outcome

- **TransactionInDoubtException**. Some of the transactional operations may have succeeded and some may have failed. This can happen to clients and to any member running a transaction on another data host. To manage this, you may want to install cache listeners in the members running the transaction code. Use the listeners to monitor and record the changes you receive from your transactions so you can recover as needed if you get this exception.

## Transaction Coding Examples

This section provides several code examples for writing and executing transactions.

## Basic Transaction Example

This example begins a transaction, updates two replicated regions, `cash` and `trades`, and commits.

If the commit fails, it throws a `CommitConflictException` and the transaction is rolled back. In this example, this type of exception could only occur if there was concurrent access to the same key and value inside another transaction.

```
Cache c = CacheFactory.create(DistributedSystem.connect(null));
AttributesFactory af = new AttributesFactory();
af.setDataPolicy(DataPolicy.REPLICATE);
Region<String, Integer> cash = c.createRegion("cash", af.create());
Region<String, Integer> trades = c.createRegion("trades", af.create());
CacheTransactionManager txmgr = c.getCacheTransactionManager();
try {
    txmgr.begin();
    final String customer = "Customer1";
    final Integer purchase = Integer.valueOf(1000);
    // Decrement cash
    Integer cashBalance = cash.get(customer);
    Integer newBalance =
        Integer.valueOf((cashBalance != null ? cashBalance : 0)
            - purchase);
    cash.put(customer, newBalance);
    // Increment trades
    Integer tradeBalance = trades.get(customer);
    newBalance =
        Integer.valueOf((tradeBalance != null ? tradeBalance : 0)
            + purchase);

    trades.put(customer, newBalance);
    txmgr.commit();
}
catch (CommitConflictException conflict)
```

## Basic Suspend and Resume Transaction Example

This example suspends and resumes a transaction.

```
CacheTransactionManager txMgr = cache.getCacheTransactionManager();
txMgr.begin();
region.put("key1", "value");
TransactionId txId = txMgr.suspend();
assert region.containsKey("key1") == false;
// do other operations that should not be
// part of a transaction
txMgr.resume(txId);
region.put("key2", "value");
txMgr.commit();
```

## Transaction Run in Multiple Regions Example

This example creates a Customer and Order partitioned region, and then updates a customer and one customer order in a transaction.

```
/**
 * An example for using GemFire Transactions.
 * Creates a Customer and Order partitioned region, and then
 * updates the customer and one customer order in a transaction.
 */
public class TransactionalPeer {
    /** The region for storing customer information*/
```

```

private Region<CustomerId, String> custRegion;
/** The region for storing order information of a customer*/
private Region<OrderId, String> orderRegion;

private static final int MAX_KEYS_IN_REGION = 10;

public TransactionalPeer(boolean isEmpty) {
    createCache();
    createRegions(isEmpty);
}

/**
 * Connects to the distributed system and creates the cache
 */
protected void createCache() {
    Properties props = new Properties();
    system = DistributedSystem.connect(props);
    cache = CacheFactory.create(system);
}

/**
 * Creates the customer and order regions
 * @param isEmpty true if these regions should not store data locally,
false otherwise
 */
protected void createRegions(boolean isEmpty) {
    PartitionAttributesFactory paf =
        new PartitionAttributesFactory<CustomerId, String>();
    paf.setPartitionResolver(new CustomerOrderResolver());
    if (isEmpty) {
        paf.setLocalMaxMemory(0);
    }
    AttributesFactory af = new AttributesFactory<CustomerId, String>();
    af.setDataPolicy(DataPolicy.PARTITION);
    af.addCacheListener(new LoggingCacheListener());
    af.setPartitionAttributes(paf.create());
    custRegion = cache.createRegion("customer", af.create());

    paf = new PartitionAttributesFactory<OrderId, String>();
    paf.setColocatedWith("customer");
    paf.setPartitionResolver(new CustomerOrderResolver());
    if (isEmpty) {
        paf.setLocalMaxMemory(0);
    }
    af = new AttributesFactory<OrderId, String>();
    af.setDataPolicy(DataPolicy.PARTITION);
    af.addCacheListener(new LoggingCacheListener());
    af.setPartitionAttributes(paf.create());
    orderRegion = cache.createRegion("order", af.create());
}

/**
 * Runs the example by asking for user input for the type of transaction
 * to be run
 * @throws IOException if there is Exception while reading user input
 */
private void runExample() throws IOException {
    System.out.println("Please start the other JVM and press enter to populate
regions");
    BufferedReader reader = new BufferedReader(new

```

```

InputStreamReader(System.in));
reader.readLine();
if (custRegion.size() == 0) {
    System.out.println("Populating region...");
    populateRegion();
    System.out.println("Complete");
} else {
    System.out.println("Regions already populated");
}
Random random = new Random();
while (true) {
    System.out.println("Press 1 to run a transaction, 2 to run a transactional
function");
    String input = reader.readLine();
    CustomerId custToUpdate = new
CustomerId(random.nextInt(MAX_KEYS_IN_REGION+1));
    OrderId orderToUpdate = new OrderId(random.nextInt(100), custToUpdate);

    if ("1".equals(input.trim())) {
        CacheTransactionManager mgr =
custRegion.getCache().getCacheTransactionManager();
        System.out.println("Starting a transaction...");
        mgr.begin();
        int randomInt = random.nextInt(1000);
        System.out.println("for customer region updating "+custToUpdate);
        custRegion.put(custToUpdate, "updatedCustomer_"+randomInt);
        System.out.println("for order region updating "+orderToUpdate);
        orderRegion.put(orderToUpdate, "newOrder_"+randomInt);
        mgr.commit();
        System.out.println("transaction completed");
    } else if ("2".equals(input.trim())) {
        System.out.println("Executing Function");
        Set filter = new HashSet();
        filter.add(custToUpdate);
        System.out.println("Invoking Function");
        //please refer to the function service documentation for more information

        FunctionService.onRegion(custRegion).withFilter(filter).withArgs(
            orderToUpdate).execute(new TransactionalFunction()).getResults();
        System.out.println("Function invocation completed");
    } else {
        continue;
    }
}

/**
 * gets us started by putting some data in the customer
 * and order regions
 */
private void populateRegion() {
    for (int i=0; i<MAX_KEYS_IN_REGION/2; i++) {
        CustomerId custId = new CustomerId(i);
        OrderId orderId = new OrderId(i, custId);
        custRegion.put(custId, "customer_"+i);
        orderRegion.put(orderId, "order_"+i);
    }
}
public static void main(String[] args) throws IOException {
    boolean isEmpty = false;
    if (args.length > 1) {

```

```

        showUsage();
        System.exit(1);
    } else if (args.length == 1) {
        if ("empty".equalsIgnoreCase(args[0])) {
            isEmpty = true;
        } else {
            showUsage();
            System.exit(1);
        }
    }
    TransactionalPeer peer = new TransactionalPeer(isEmpty);
    peer.runExample();
}

/**
 * prints the usage
 */
private static void showUsage() {
    System.out.println("java TransactionalPeer [empty]");
}
}

```

### Transactional Using a Function Example

This example demonstrates a function that does transactional updates to Customer and Order regions.

```

/**
 * This function does transactional updates to customer and order regions
 */
public class TransactionalFunction extends FunctionAdapter {

    private Random random = new Random();
    /* (non-Javadoc)
     * @see com.gemstone.gemfire.cache.execute.FunctionAdapter#
     * execute(com.gemstone.gemfire.cache.execute.FunctionContext)
     */
    @Override
    public void execute(ExecutionContext context) {
        RegionFunctionContext rfc = (RegionFunctionContext)context;
        Region<CustomerId, String> custRegion = rfc.getDataSet();
        Region<OrderId, String>
        orderRegion = custRegion.getCache().getRegion("order");
        CacheTransactionManager
        mgr = custRegion.getCache().getCacheTransactionManager();
        CustomerId custToUpdate = (CustomerId)rfc.getFilter().iterator().next();

        OrderId orderToUpdate = (OrderId)rfc.getArguments();
        System.out.println("Starting a transaction...");
        mgr.begin();
        int randomInt = random.nextInt(1000);
        System.out.println("for customer region updating "+custToUpdate);
        custRegion.put(custToUpdate,
        "updatedCustomer_"+custToUpdate.getCustId()+"_"+randomInt);
        System.out.println("for order region updating "+orderToUpdate);
        orderRegion.put(orderToUpdate,
        "newOrder_"+orderToUpdate.getOrderID()+"_"+randomInt);
        mgr.commit();
        System.out.println("transaction completed");
        context.getResultSender().lastResult(Boolean.TRUE);
    }
}

```

```
/* (non-Javadoc)
 * @see com.gemstone.gemfire.cache.execute.FunctionAdapter#getId()
 */
@Override
public String getId() {
    return "TxFunction";
}

}
```

### GemFire JTA Transaction Example

This example shows how to run a JTA global transaction using GemFire as the JTA transaction manager.

The external data sources used in this transaction are configured in cache.xml. See [Configuring Database Connections Using JNDI](#) on page 404 for examples.

```
Region r = ...; // the gemfire regionDataSource
ds = ...; // other datasource

try {
    Context ctx = cache.getJNDIContext();
    Connection conn = null;
    UserTransaction
    tx = (UserTransaction)
    ctx.lookup("java:/UserTransaction");
    tx.begin();
    conn = ds.getConnection();
    Statement stmt = conn.createStatement();
    String sqlSTR = "insert into " + tableName + " values (.....
) ";
    stmt.executeUpdate(sqlSTR);
    r.put("key", "value");
    stmt.close();
    tx.commit();
    conn.close();
} catch (NamingException e) {
    // handle the exception
}
```



# Chapter 38

---

## Function Execution

This section provides details about how to write, register, and execute functions for particular use cases.

### Use Cases for Function Execution

The function execution service provides solutions for these application use cases:

- An application that executes a server-side transaction or carries out data updates using the GemFire distributed locking service.
- An application that needs to initialize some of its components once on each server, which might be used later by executed functions.
- Initialization and startup of a third-party service, such as a messaging service.
- Any arbitrary aggregation operation that requires iteration over local data sets that can be done more efficiently through a single call to the cache server.
- Any kind of external resource provisioning that can be done by executing a function on a server.

### How Function Execution Works

#### Where Functions Are Executed

You can execute data-independent functions or data-dependent functions in vFabric GemFire. You run data-independent functions by targeting a specific member or specific members in your distributed system, or by targeting logical member groups on which to execute your function. If you are executing a data-dependent function, you specify a region on which to execute the function.

- On a specific member or members. Execute the function in your own distributed system, and specify the member or members where you want to run the function.
- On a specific server or set of servers. If you are connected to a distributed system as a client, you can execute the function on a server or servers configured for a specific connection pool, or on a server or servers connected to a given cache.
- On member groups or on a single member within each member group. In GemFire 7.0, you can organize members into logical member groups. (See [Using Member Groups](#) on page 745 for more information.) You can invoke a data independent function on all members in a specified member group (or member groups) or execute the function on only one member of each specified member group.
- On a data set. Specify a region and possibly a set of keys on which to run the function.

See the `com.gemstone.gemfire.cache.execute.FunctionService` Java API documentation for more details.

## How Functions Are Executed

The following things occur when a function is executed:

1. When you call the `execute` method on the `Function Execution` object, the execution is sent to all members where it needs to run. The locations are determined by the `FunctionService on*` method calls, region configuration, and any filters. With `onRegion`, for a client region, the function is sent to the server system. For a non-client region, the function is handled in the calling application's distributed system.
2. If the function has results, the result is returned to the `execute` method call in a `ResultCollector` object.
3. The originating member collects results using `ResultCollector.getResult`.

## Highly Available Functions

Generally, function execution errors are returned to the calling application. You can code for high availability for `onRegion` functions that return a result, so GemFire automatically retries a function if it does not execute successfully. To be highly available, the function must be coded and configured for it, and the calling application must invoke the function using the results collector `getResult` method.

When a failure occurs (such as an execution error or member crash while executing), the system responds as follows:

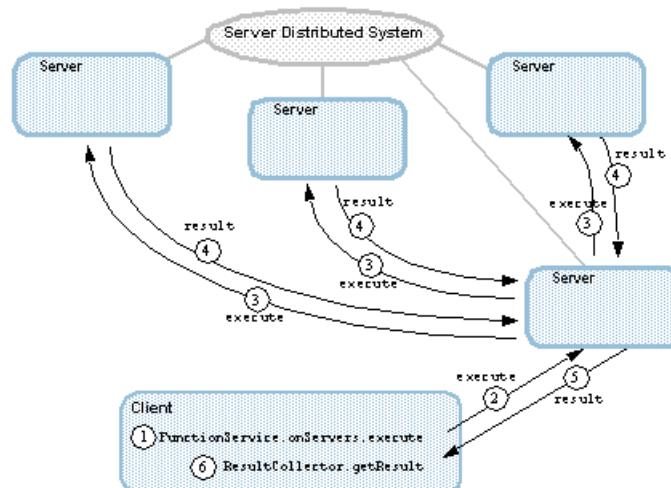
1. Waits for all calls to return
2. Sets a boolean indicating a re-execution is being done
3. Calls the result collector's `clearResults` method
4. Executes the function

For client regions, the system retries the execution according to `com.gemstone.gemfire.cache.client.Pool retryAttempts`. If the function fails to run every time, the final exception is returned to the `getResult` method.

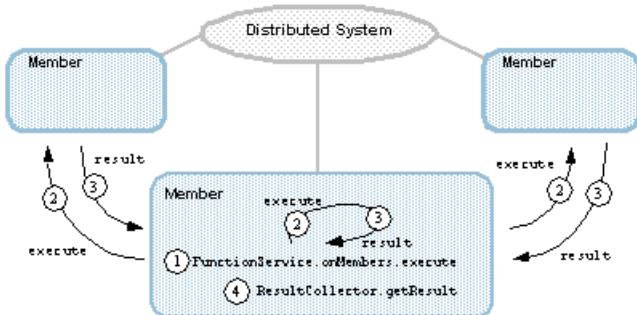
For member calls, the system retries until it succeeds or no data remains in the system for the function to operate on.

## Function Execution Scenarios

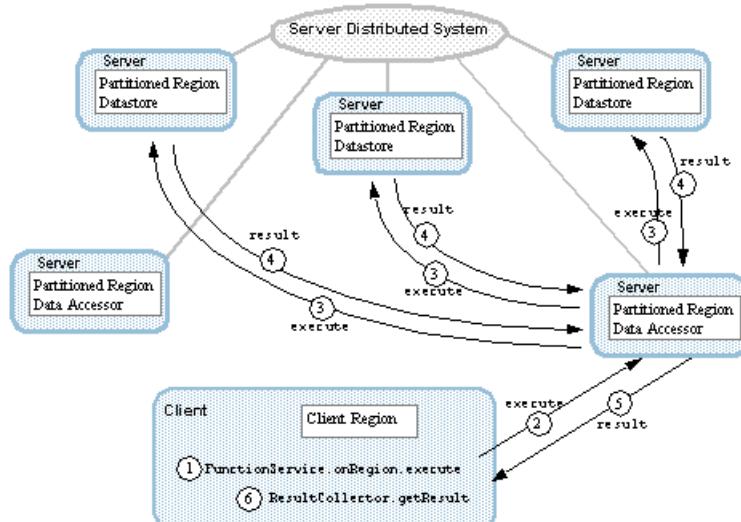
The diagram shows the sequence of events for a data-independent function run on a client's server. The function is executed against all available servers:



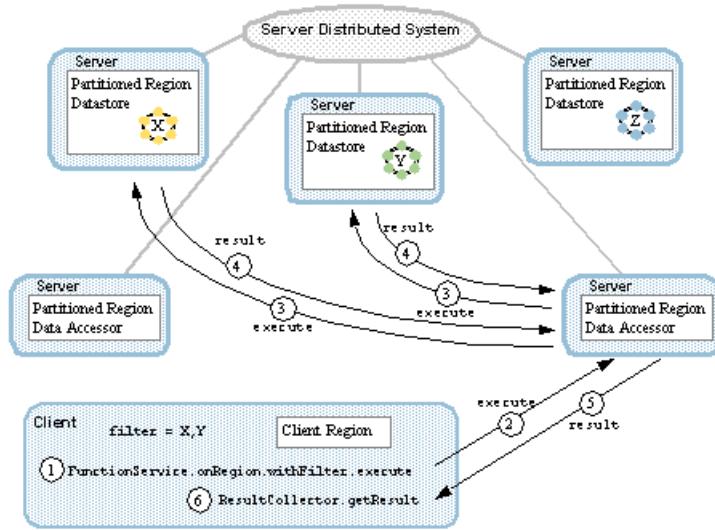
The next figure shows the sequence of events for a data-independent function executed against all available members in the calling application's distributed system:



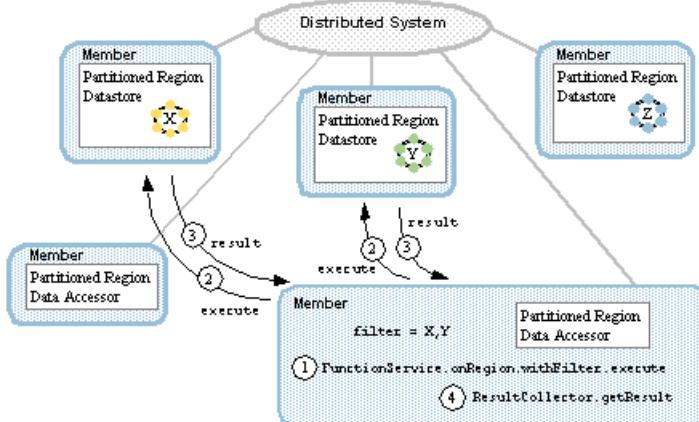
The next figure shows a data-dependent function run on a client region. The client region is connected to the server system, where the region is partitioned, so the function automatically goes to the server system to run against all servers holding data for the region.



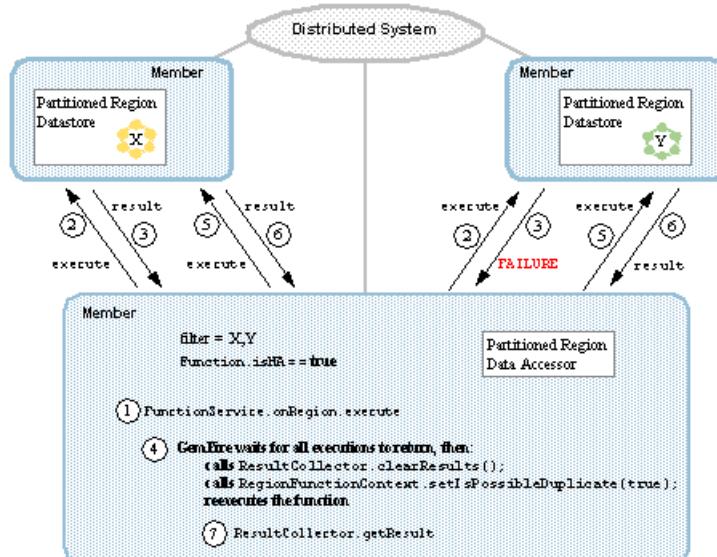
The next figure shows the same data-dependent function with the added specification of a set of keys on which to run. Servers that do not hold any keys are left out of the function execution.



The next figure shows a data-dependent call is on a region that is not configured as a client region, so it runs in the caller's distributed system.



The next figure demonstrates a sequence of steps in a call to a highly available function. The call fails the first time on one of the participating members and is successfully run a second time on all members.



## Executing a Function in vFabric GemFire

In this procedure it is assumed that you have your members and regions defined where you want to run functions.

Main tasks:

1. Write the function code.
2. Register the function on all servers where you want to execute the function. The easiest way to register a function is to use the gfsh deploy command to deploy the JAR file containing the function code. Deploying the JAR automatically registers the function for you. See [Register the Function Automatically by Deploying a JAR](#) on page 423 for details. Alternatively, you can write the XML or application code to register the function. See [Register the Function Programmatically](#) on page 423 for details.
3. Write the application code to run the function and, if the function returns results, to handle the results.
4. If your function returns results and you need special results handling, code a custom `ResultsCollector` implementation and use it in your function execution.

### Write the Function Code

To write the function code, you implement the `Function` interface or extend the `FunctionAdapter` class. Both are in the `com.gemstone.gemfire.cache.execute` package. The adapter class provides some default implementations for methods, which you can override.

Code the methods you need for the function. These steps do not have to be done in this order.

1. Code `getId` to return a unique name for your function. You can use this name to access the function through the `FunctionService` API.
2. For high availability:
  - a. Code `isHA` to return true to indicate to GemFire that it can re-execute your function after one or more members fails
  - b. Code your function to return a result
  - c. Code `hasResult` to return true

3. Code `hasResult` to return true if your function returns results to be processed and false if your function does not return any data - the fire and forget function. `FunctionAdapter hasResult` returns true by default.
4. If the function will be executed on a region, code `optimizeForWrite` to return false if your function only reads from the cache, and true if your function updates the cache. The method only works if, when you are running the function, the `Execution` object is obtained through a `FunctionService onRegion` call. `FunctionAdapter optimizeForWrite` returns false by default.
5. Code the `execute` method to perform the work of the function.
  - a. Make `execute` thread safe to accommodate simultaneous invocations.
  - b. For high availability, code `execute` to accommodate multiple identical calls to the function. Use the `RegionFunctionContext isPossibleDuplicate` to determine whether the call may be a high-availability re-execution. This boolean is set to true on execution failure and is false otherwise.



**Note:** The `isPossibleDuplicate` boolean can be set following a failure from another member's execution of the function, so it only indicates that the execution might be a repeat run in the current member.

- c. Use the function context to get information about the execution and the data:
  - The context holds the function ID, the `ResultSender` object for passing results back to the originator, and function arguments provided by the member where the function originated.
  - The context provided to the function is the `FunctionContext`, which is automatically extended to `RegionFunctionContext` if you get the `Execution` object through a `FunctionService onRegion` call.
  - For data dependent functions, the `RegionFunctionContext` holds the `Region` object, the `Set` of key filters, and a boolean indicating multiple identical calls to the function, for high availability implementations.
  - For partitioned regions, the `PartitionRegionHelper` provides access to additional information and data for the region. For single regions, use `getLocalDataForContext`. For colocated regions, use `getLocalColocatedRegions`.

Example function code:

```
package quickstart;

import java.io.Serializable;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

import com.gemstone.gemfire.cache.execute.FunctionAdapter;
import com.gemstone.gemfire.cache.execute.FunctionContext;
import com.gemstone.gemfire.cache.execute.RegionFunctionContext;
import com.gemstone.gemfire.cache.partition.PartitionRegionHelper;

public class MultiGetFunction extends FunctionAdapter {

    public void execute(FunctionContext fc) {
        RegionFunctionContext context = (RegionFunctionContext)fc;
        Set keys = context.getFilter();
        Set keysTillSecondLast = new HashSet();
        int setSize = keys.size();
        Iterator keysIterator = keys.iterator();
        for(int i = 0; i < (setSize -1); i++)
        {
            if(keysTillSecondLast.contains(keysIterator.next()))
            {
                keysTillSecondLast.remove(keysIterator.previous());
            }
            else
            {
                keysTillSecondLast.add(keysIterator.next());
            }
        }
        if(keysTillSecondLast.size() > 0)
        {
            context.getResultSender().sendResult(keysTillSecondLast);
        }
    }
}
```

```

        keysTillSecondLast.add(keysIterator.next());
    }
    for (Object k : keysTillSecondLast) {
        context.getResultSender().sendResult(
            (Serializable)PartitionRegionHelper.getLocalDataForContext(context)

                .get(k));
    }
    Object lastResult = keysIterator.next();
    context.getResultSender().lastResult(
        (Serializable)PartitionRegionHelper.getLocalDataForContext(context)

            .get(lastResult));
}

public String getId() {
    return getClass().getName();
}
}

```

## Register the Function Automatically by Deploying a JAR

When you deploy a JAR file that contains a Function (in other words, contains a class that implements the Function interface), the Function will be automatically registered via the `FunctionService.registerFunction` method.

To register a function by using gfsh:

1. Package your class files into a JAR file.
2. Start a gfsh prompt. If necessary, start a Locator and connect to the GemFire distributed system where you want to run the function.
3. At the gfsh prompt, type the following command:

```
gfsh>deploy --jar=group1_functions.jar
```

where group1\_functions.jar corresponds to the JAR file that you created in step 1.

If another JAR file is deployed (either with the same JAR filename or another filename) with the same Function, the new implementation of the Function will be registered, overwriting the old one. If a JAR file is undeployed, any Functions that were auto-registered at the time of deployment will be unregistered. Since deploying a JAR file that has the same name multiple times results in the JAR being un-deployed and re-deployed, Functions in the JAR will be unregistered and re-registered each time this occurs. If a Function with the same ID is registered from multiple differently named JAR files, the Function will be unregistered if either of those JAR files is re-deployed or un-deployed.

See [Deploying Application JARs to vFabric GemFire Members](#) on page 747 for more details on deploying JAR files.

## Register the Function Programmatically

This section applies to functions that are invoked using the `Execution.execute(String functionId)` signature. When this method is invoked, the calling application sends the function ID to all members where the `Function.execute` is to be run. Receiving members use the ID to look up the function in the local `FunctionService`. In order to do the lookup, all of the receiving member must have previously registered the function with the function service.

The alternative to this is the `Execution.execute(Function function)` signature. When this method is invoked, the calling application serializes the instance of `Function` and sends it to all members where the `Function.execute` is to be run. Receiving members deserialize the `Function` instance, create a new local

instance of it, and run execute from that. This option is not available for non-Java client invocation of functions on servers.

Your Java servers must register functions that are invoked by non-Java clients. You may want to use registration in other cases to avoid the overhead of sending Function instances between members.

Register your function using one of these methods:

- XML:

```
<cache>
  ...
  </region>
<function-service>
  <function>

    <class-name>com.bigFatCompany.tradeService.cache.func.TradeCalc</class-name>

  </function>
</function-service>
```

- Java:

```
myFunction myFun = new myFunction();
FunctionService.registerFunction(myFun);
```



**Note:** Modifying a function instance after registration has no effect on the registered function. If you want to execute a new function, you must register it with a different identifier.

## Run the Function

This assumes you've already followed the steps for writing and registering the function.

In every member where you want to explicitly execute the function and process the results, you can use the gfsh command line to run the function or you can write an application to run the function.

### Running the Function Using gfsh

1. Start a gfsh prompt.
2. If necessary, start a Locator and connect to the GemFire distributed system where you want to run the function.
3. At the gfsh prompt, type the following command:

```
gfsh> execute function --id=function_id
```

Where *function\_id* equals the unique ID assigned to the function. You can obtain this ID using the `Function.getId` method.

See [Function Execution Commands](#) on page 663 for more gfsh commands related to functions.

### Running the Function via API Calls

1. Use one of the `FunctionService.on*` methods to create an `Execute` object. The `on*` methods, `onRegion`, `onMembers`, etc., define the highest level where the function is run. For colocated partitioned regions, use `onRegion` and specify any one of the colocated regions. The function run using `onRegion` is referred to as a data dependent function - the others as data-independent functions.
2. Use the `Execution` object as needed for additional function configuration. You can:
  - Provide a key `Set` to `withFilters` to narrow the execution scope for `onRegion` `Execution` objects. You can retrieve the key set in your `Function execute` method through `RegionFunctionContext.getFilter`.

- Provide function arguments to `withArgs`. You can retrieve these in your Function `execute` method through `FunctionContext.getArguments`.
  - Define a custom `ResultCollector`
3. Call the `Execution` object to `execute` method to run the function.
  4. If the function returns results, call `getResult` from the results collector returned from `execute` and code your application to do whatever it needs to do with the results.



**Note:** For high availability, you must call the `getResult` method.

Example of running the function - for executing members:

```
MultiGetFunction function = new MultiGetFunction();
FunctionService.registerFunction(function);

writeToStdout("Press Enter to continue.");
stdinReader.readLine();

Set keysForGet = new HashSet();
keysForGet.add("KEY_4");
keysForGet.add("KEY_9");
keysForGet.add("KEY_7");

Execution execution = FunctionService.onRegion(exampleRegion)
    .withFilter(keysForGet)
    .withArgs(Boolean.TRUE)
    .withCollector(new MyArrayListResultCollector());

ResultCollector rc = execution.execute(function);
// Retrieve results, if the function returns results
List result = (List)rc.getResult();
```

## Write a Custom Results Collector

This topic applies to functions that return results.

When you execute a function that returns results, the function stores the results into a `ResultCollector` and returns the `ResultCollector` object. The calling application can then retrieve the results through the `ResultCollector getResult` method. Example:

```
ResultCollector rc = execution.execute(function);
List result = (List)rc.getResult();
```

GemFire's default `ResultCollector` collects all results into an `ArrayList`. Its `getResult` methods block until all results are received. Then they return the full result set.

To customize results collecting:

1. Write a class that extends `ResultCollector` and code the methods to store and retrieve the results as you need. Note that the methods are of two types:
  - a. `addResult` and `endResults` are called by GemFire when results arrive from the `Function` instance `SendResults` methods
  - b. `getResult` is available to your executing application (the one that calls `Execution.execute`) to retrieve the results
2. Use high availability for `onRegion` functions that have been coded for it:
  - a. Code the `ResultCollector clearResults` method to remove any partial results data. This readies the instance for a clean function re-execution.

- b. When you invoke the function, call the result collector `getResults` method. This enables the high availability functionality.
3. In your member that calls the function execution, create the `Execution` object using the `withCollector` method, and passing it your custom collector. Example:

```
Execution execution = FunctionService.onRegion(exampleRegion)
    .withFilter(keysForGet)
    .withArgs(Boolean.TRUE)
    .withCollector(new MyArrayListResultCollector());
```

### **Targeting Single Members of a Member Group or Entire Member Groups**

To execute a data independent function on a group of members or one member in a group of members, you can write your own nested function. You will need to write one nested function if you are executing the function from client to server and another nested function if you are executing a function from server to all members.

# Part 7

---

## Managing vFabric GemFire

*Managing vFabric GemFire* describes how to plan and implement tasks associated with managing, monitoring, and troubleshooting vFabric GemFire.

### Topics:

- GemFire Management and Monitoring
- Heap Use and Management
- Disk Storage
- Cache and Region Snapshots
- Network Partitioning
- Security
- Performance Tuning and Configuration
- Statistics
- Troubleshooting and System Recovery



# Chapter 39

---

## GemFire Management and Monitoring

vFabric GemFire provides APIs and tools for managing your distributed system and monitoring the health of your distributed system members.

### Management and Monitoring Features

Beginning in the GemFire 7.0 release, GemFire uses a federated Open MBean strategy to manage and monitor all members of the distributed system. This strategy gives you a consolidated, single-agent view of the distributed system.

Application and manager development is much easier because you do not have to find the right MBeanServer to make a request on an MBean. Instead, you interact with a single MBeanServer that aggregates MBeans from all other local and remote MBeanServers.

Some other key advantages and features of GemFire 7.0's new administration architecture:

- GemFire monitoring is tightly integrated into GemFire's processes instead of running in a separately installed and configured monitoring agent. You can use the same framework to actually manage GemFire and perform administrative operations, not just monitor it.
- All GemFire MBeans are now *MXBeans*. They represent useful and relevant information on the state of the distributed system and all its members. Because MXBeans use the Open MBean model with a predefined set of types, clients and remote management programs no longer require access to model-specific classes representing your MBean types. Using MXBeans adds flexibility to your selection of clients and makes the GemFire management and monitoring much easier to use.
- Each member in the distributed system is manageable through MXBeans, and each member hosts its own MXBeans in a Platform MBeanServer.
- Any GemFire member can be configured to provide a federated view of all the MXBeans for all members in a GemFire cluster.
- GemFire has also modified its use of JMX to be industry-standard and friendly to generic JMX clients. You can now easily monitor or manage the distributed system by using any third-party tool that is compliant with JMX. For example, JConsole.

### References

For more information on MXBeans and Open MBeans, see:

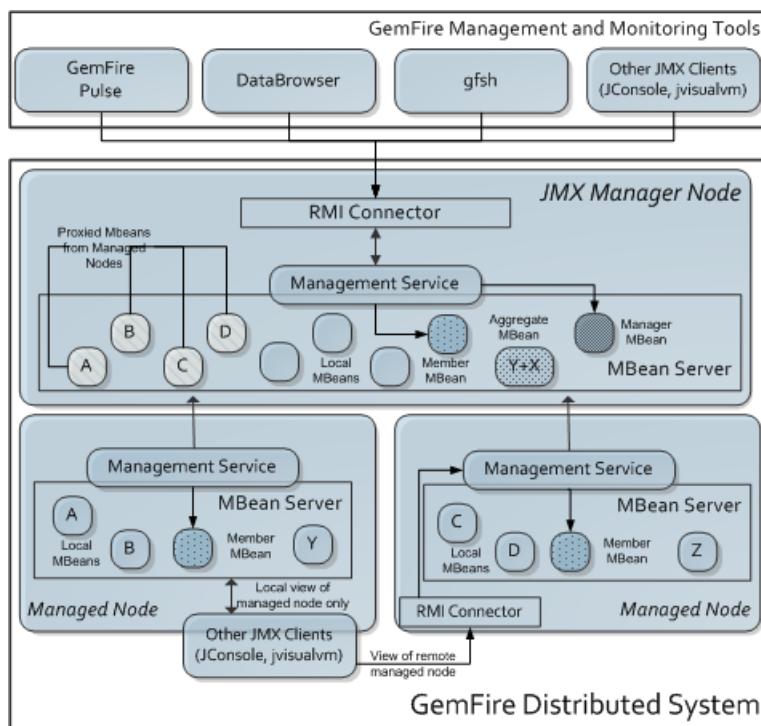
- <http://docs.oracle.com/javase/6/docs/api/javax/management/MXBean.html>
- <http://docs.oracle.com/javase/6/docs/api/javax/management/openmbean/package-summary.html>

## Architecture and Components

GemFire's management and monitoring system consists of one JMX Manager node (there should only be one) and one or more managed nodes within a distributed system. All members in the distributed system are manageable through MBeans and GemFire Management Service APIs.

### Architecture

The following diagram depicts the architecture of the management and monitoring system components.



In this architecture every GemFire member is manageable. All GemFire MBeans for the local GemFire processes are automatically registered in the Platform MBeanServer (the default MBeanServer of each JVM that hosts platform MXBeans.)

### Managed Node

Each member of a distributed system is a managed node. Any node that is not currently also acting as a JMX Manager node is referred to simply as a managed node. A managed node has the following resources so that it can answer JMX queries both locally and remotely:

- An RMI connector that allows a remote JMX client to connect directly to the node and access all local MXBeans.
- Local MXBeans that represent the locally monitored components on the node. Possible MXBeans (depending on the node's functionality include):
  - MemberMXBean
  - RegionMXBean
  - CacheServerMXBean
  - LockServiceMXBean
  - DiskStoreMXBean
  - GatewaySenderMXBean
  - GatewayReceiverMXBean

- LocatorMXBean
- Built-in platform MBeans

### **JMX Manager Node**

A JMX Manager node is a member that can manage other GemFire members --that is, other managed nodes -- as well as itself. A JMX Manager node can manage all other members in the distributed system.

To convert a managed node to a JMX Manager node, you configure the GemFire property `jmx-manager=true`, in the `gemfire.properties` file, and start the member as a JMX Manager node. You start the member as a JMX Manager node when you provide `--J=-Dgemfire.jmx-manager=true` as an argument to either the `start server` or `start locator` command. See [Starting a JMX Manager](#) on page 433 for more information.

The JMX Manager node has the following extra resources allocated so that it can answer JMX queries:

- RMI connector that allows JMX clients to connect to and access all MXBeans in the distributed system.
- Local MXBeans that represent the locally monitored components on this node, same as any other managed node.
- Aggregate MXBeans:
  - DistributedSystemMXBean.
  - DistributedRegionMXBean.
  - MemberGroupMXBean.
  - DistributedLockServiceMXBean
- ManagerMXBean with Scope=ALL, which allows various distributed system-wide operations.
- Proxy to MXBeans on managed nodes.
- Built-in platform MXBeans.

### **JMX Integration**

Management and monitoring tools such as the GemFire SHell (gfsh) and GemFire Pulse use JMX/RMI as the communication layer to connect to GemFire nodes. All GemFire processes by default allow JMX connections to the Platform MBeanServer from localhost. By default, both managed nodes and JMX manager nodes have RMI connectors enabled to allow JMX client connections.

JConsole (and other similar JMX clients that support Sun's Attach API) can connect to any local JVM without requiring an RMI connector by using the Attach API. This allows connections from the same machine.

JConsole (and other JMX clients) can connect to any JVM if that JVM is configured to start an RMI connector. This allows remote connections from other machines.

JConsole can connect to any GemFire member, but if it connects to a non-JMX-Manager member, JConsole only detects the local MBeans for the node, and not MBeans for the cluster.

When a GemFire locator or server becomes a JMX Manager for the cluster, it enables the RMI connector. JConsole can then connect only to that one JVM to view the MBeans for the entire cluster. It does not need to connect to all the other JVMs. GemFire manages the inter-JVM communication required to provide a federated view of all MBeans in the distributed system.

GemFire SHell (gfsh) can only connect to a JMX Manager or to a locator. If connected to a locator, the locator provides the necessary connection information for the existing JMX Manager. If the locator detects a JMX Manager is not already running in the cluster, the locator makes itself a JMX Manager. gfsh cannot connect to other non-Manager or non-locator members.

For information on how to configure the RMI registry and RMI connector, see [Configuring RMI Registry Ports and RMI Connectors](#) on page 446.

## Management APIs

GemFire management APIs represent the Gemfire cluster to a JMX user. However, they do not provide functionality that is otherwise present in JMX. They only provide a gateway into various services exclusively offered by GemFire monitoring and management.

The entry point to GemFire management is through the ManagementService interface. For example, to create an instance of the Management Service:

```
ManagementService service = ManagementService.getManagementService(cache);
```

The resulting ManagementService instance is specific to the provided cache and its distributed system. The implementation of getManagementService is a singleton for now but may eventually support multiple cache instances.

You can use the GemFire management APIs to accomplish the following tasks:

- Monitor the health status of clients.
- Obtain the status and results of individual disk backups.
- View metrics related to disk usage and performance for a particular member.
- Browse GemFire properties set for a particular member.
- View JVM metrics such as memory, heap, and thread usage.
- View network metrics, such as bytes received and sent
- View partition region attributes such as total number of buckets, redundant copy, and maximum memory information.
- View persistent member information such as disk store ID.
- Browse region attributes.

See the JavaDocs for the `com.gemstone.gemfire.management` package for more details.

You can also execute gfsh commands using the ManagementService API. See [Executing gfsh Commands through the Management API](#) on page 446 and the JavaDocs for the `com.gemstone.gemfire.management.cli` package.

## GemFire Management and Monitoring Tools

This section lists the currently available tools for managing and monitoring GemFire:

- **gfsh.** vFabric GemFire command-line interface that provides a simple & powerful command shell that supports the administration, debugging and deployment of GemFire applications. It features context sensitive help, scripting and the ability to invoke any commands from within the application using a simple API. See [Using the GemFire SHell \(gfsh\)](#) on page 641.
- **GemFire Pulse.** Easy-to-use, browser-based dashboard for monitoring GemFire deployments. GemFire Pulse provides an integrated view of all GemFire members within a distributed system, as well as in remote distributed systems that are connected over a WAN. See [GemFire Pulse](#) on page 819.
- **DataBrowser.** Graphical interface for performing OQL ad-hoc queries in a GemFire distributed system. See [DataBrowser](#) on page 827.
- **Other Java Monitoring Tools such as JConsole and jvisualvm.** JConsole is a JMX-based management and monitoring tool provided in the Java 2 Platform that provides information on the performance and consumption of resources by Java applications. See <http://docs.oracle.com/javase/6/docs/technotes/guides/management/jconsole.html>. **Java VisualVM (jvisualvm)** is a profiling tool for analyzing your Java Virtual Machine. Java VisualVM is useful to Java application developers to troubleshoot applications and to monitor and improve the applications' performance. Java VisualVM can allow developers to generate and analyse heap dumps, track down memory leaks, perform and monitor garbage collection, and perform lightweight memory and CPU profiling. For more details on using jvisualvm, see <http://docs.oracle.com/javase/6/docs/technotes/tools/share/jvisualvm.html>.

## JMX Manager Operations

Any GemFire member can host an embedded JMX Manager, which provides a federated view of all MBeans for the distributed system. The member can be configured to be a manager at startup or anytime during its life by invoking the appropriate API calls on the ManagementService.

You need to have a JMX Manager started in your distributed system in order to use GemFire management and monitoring tools such as [Using the GemFire SHell \(gfsh\)](#) on page 641 and [GemFire Pulse](#) on page 819.

### Starting a JMX Manager

JMX Manager nodes are members that manage other GemFire members (as well as themselves). A JMX Manager node can manage all other members in the distributed system. Typically a locator will function as the JMX Manager, but you can also turn any other distributed system member such as a server into a JMX Manager node as well.

To allow a server to become a JMX Manager you configure GemFire property `jmx-manager=true`, in the `server'sgemfire.properties` file. This property configures the node to become a JMX Manager node passively; if gfsh cannot locate a JMX Manager when connecting to the distributed system, the server node will be started as a JMX Manager node.



**Note:** The default property setting for all locators is `gemfire.jmx-manager=true`. For other members, the default property setting is `gemfire.jmx-manager=false`.

To force a server to become a JMX Manager node whenever it is started, set the GemFire properties `jmx-manager-start=true` and `jmx-manager=true` in the server's `gemfire.properties` file. Note that both of these properties must be set to true for the node.

To start the member as a JMX Manager node on the command line, provide

`--J=-Dgemfire.jmx-manager-start=true` and `--J=-Dgemfire.jmx-manager=true` as arguments to either the `start server` or `start locator` command.

For example, to start a server as a JMX Manager on the gfsh command line:

```
gfsh>start server --name=<server-name> --J=-Dgemfire.jmx-manager=true \
--J=-Dgemfire.jmx-manager-start=true
```

By default, any locator can become a JMX Manager when started. When you start up a locator, if no other JMX Manager is detected in the distributed system, the locator starts one automatically. If you start a second locator, it will detect the current JMX Manager and will not start up another JMX Manager unless the second locator's `gemfire.jmx-manager-start` property is set to true.

For most deployments, you only need to have one JMX Manager per distributed system. However, you can run more than JMX Manager if necessary. If you want to provide high-availability and redundancy for the Pulse monitoring tool, or if you are running additional JMX clients other than gfsh, then use the `jmx-manager-start=true` property to force individual nodes (either locators or servers) to become JMX Managers at startup. Since there is some performance overhead to being a JMX Manager, we recommend using locators as JMX Managers.

See [start server](#) on page 722 and [start locator](#) on page 720 for more details.

After the node becomes a JMX Manager, all other `jmx-manager-*` configuration properties listed in [Configuring a JMX Manager](#) on page 435 are applied.

The following is an example of starting a new locator that also starts an embedded JMX Manager (after detecting that another JMX Manager does not exist). In addition, gfsh also automatically connects you to the new JMX Manager. For example:

```
gfsh>start locator --name=locator1
Starting a Locator in C:\vFabricGemFire70\Latest\locator1 on
```

```

GemFireStymon[10334]
as locator1...
.
.
.
Locator in C:\vFabricGemFire70\Latest\locator1 on GemFireStymon[10334] as
locator1 is currently online.
Process ID: 3936
Uptime: 17 seconds
GemFire Version: 7.0
Java Version: 1.6.0_26
Log File: C:\vFabricGemFire70\Latest\locator1\locator1.log
JVM Arguments: -Dgemfire.launcher.registerSignalHandlers=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: ;C:\vFabricGemFire70\Latest\lib\gfsh-dependencies.jar;
C:\vFabricGemFire70\Latest\lib\gemfire.jar;C:\vFabricGemFire70\Latest\lib\antlr.jar;
C:\vFabricGemFire70\Latest\lib\mail.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-core.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-logging-juli.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-jasper.jar;
C:\vFabricGemFire70\Latest\lib\ecj-3.7.2.jar;
C:\vFabricGemFire70\Latest\lib\spring-asm-3.1.1.RELEASE.jar;
C:\vFabricGemFire70\Latest\lib\gemfire.jar;
C:\vFabricGemFire70\Latest\lib\antlr.jar;
C:\vFabricGemFire70\Latest\lib\gfsh-dependencies.jar;
C:\vFabricGemFire70\Latest\lib\gfSecurityImpl.jar;
C:\vFabricGemFire70\Latest\SampleCode\tutorial\classes;
C:\vFabricGemFire70\Latest\SampleCode\quickstart\classes;
C:\Program Files\Java\jdk1.6.0_26\lib\tools.jar

Successfully connected to: [host=GemFireStymon, port=1099]

```

Or you can enter the command directly in your terminal:

```

c:\vFabricGemFire70\Latest>gfsh start locator --name=locator1
.
.
.
Locator in C:\vFabricGemFire70\Latest\locator1 on GemFireStymon[10334] as
locator1 is currently online.
Process ID: 2688
Uptime: 17 seconds
GemFire Version: 7.0
Java Version: 1.6.0_26
Log File: C:\vFabricGemFire70\Latest\locator1\locator1.log
JVM Arguments: -Dgemfire.launcher.registerSignalHandlers=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: ;C:\vFabricGemFire70\Latest\lib\gfsh-dependencies.jar;
C:\vFabricGemFire70\Latest\lib\gemfire.jar;C:\vFabricGemFire70\Latest\lib\antlr.jar;
C:\vFabricGemFire70\Latest\lib\mail.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-core.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-logging-juli.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-jasper.jar;
C:\vFabricGemFire70\Latest\lib\ecj-3.7.2.jar;
C:\vFabricGemFire70\Latest\lib\spring-asm-3.1.1.RELEASE.jar;
C:\vFabricGemFire70\Latest\lib\gemfire.jar;
C:\vFabricGemFire70\Latest\lib\antlr.jar;
C:\vFabricGemFire70\Latest\lib\gfsh-dependencies.jar;
C:\vFabricGemFire70\Latest\lib\gfSecurityImpl.jar;
C:\vFabricGemFire70\Latest\SampleCode\tutorial\classes;
C:\vFabricGemFire70\Latest\SampleCode\quickstart\classes;
C:\Program Files\Java\jdk1.6.0_26\lib\tools.jar

Successfully connected to: [host=GemFireStymon, port=1099]

```

Locators also keep track of all nodes that can become a JMX Manager.

Immediately after creating its cache, the JMX Manager node begins federating the MBeans from other members. After the JMX Manager node is ready, the JMX Manager node sends a notification to all other members informing them that it is a new JMX Manager. The other members then put complete MBean states for themselves into each of their hidden management regions.

At any point, you can determine whether a node is a JMX Manager by using the MemberMXBean `isManager` method.

Using the Java API, any managed node that has been configured with `jmx-manager=true` can also be turned into a JMX Manager Node by invoking the `ManagementService startManager` method.



**Note:** If you start the JMX Manager programmatically and wish to enable command processing, you must also add the absolute path of `gfsh-dependencies.jar` (located in `$GEMFIRE/lib` of your GemFire installation) to the CLASSPATH of your application. Do not copy this library to your CLASSPATH because this library refers to other dependencies in `$GEMFIRE/lib` by a relative path.

## Configuring a JMX Manager

In the `gemfire.properties` file, you configure a JMX manager as follows.

Property	Description	Default
<code>jmx-manager</code>	If <code>true</code> then this member can become a JMX Manager. All other <code>jmx-manager-*</code> properties are used when it does become a JMX Manager. If this property is <code>false</code> then all other <code>jmx-manager-*</code> properties are ignored.  The default value is <code>true</code> on locators.	<code>false</code> (with Locator exception)
<code>jmx-manager-access-file</code>	By default the JMX Manager allows full access to all MBeans by any client. If this property is set to the name of a file, then it can restrict clients to only reading MBeans; they cannot modify MBeans. The access level can be configured differently in this file for each user name defined in the password file. For more information about the format of this file see Oracle's documentation of the <code>com.sun.management.jmxremote.access.file</code> system property. Ignored if <code>jmx-manager</code> is <code>false</code> or if <code>jmx-manager-port</code> is zero. See <a href="#">Setting Up JMX Authentication for GemFire Management and Monitoring</a> on page 504 for more information.	<code>not set</code>
<code>jmx-manager-bind-address</code>	By default, the JMX Manager when configured with a port listens on all the local host's addresses. You can use this property to configure which particular IP address or host name the JMX Manager will listen on. This property is ignored if <code>jmx-manager</code> is <code>false</code> or <code>jmx-manager-port</code> is zero. This address also applies to the GemFire Pulse server if you are hosting a Pulse web application.	<code>not set</code>
<code>jmx-manager-hostname-for-clients</code>	Hostname given to clients that ask the locator for the location of a JMX Manager. By default the IP address of the JMX Manager is used. However, for clients on a different network, you can configure a different hostname to be given to clients. Ignored if <code>jmx-manager</code> is <code>false</code> or if <code>jmx-manager-port</code> is zero.	<code>not set</code>
<code>jmx-manager-http-port</code>	If non-zero, when started the JMX Manager will also start an embedded Web server and will listen on this port. The Web server is used to host the GemFire Pulse Web application. If you are hosting the Pulse web app in your own Web server, then disable this embedded server by setting this property to zero. Ignored if <code>jmx-manager</code> is <code>false</code> .	8080

Property	Description	Default
jmx-manager-password-file	By default the JMX Manager allows clients without credentials to connect. If this property is set to the name of a file, only clients that connect with credentials that match an entry in this file will be allowed. Most JVMs require that the file is only readable by the owner. For more information about the format of this file see Oracle's documentation of the com.sun.management.jmxremote.password.file system property. Ignored if jmx-manager is false or if jmx-manager-port is zero. See <a href="#">Setting Up JMX Authentication for GemFire Management and Monitoring</a> on page 504 for more information.	not set
jmx-manager-port	Port on which this JMX Manager listens for client connections. If this property is set to zero, GemFire does not allow remote client connections. Alternatively, use the standard system properties supported by the JVM for configuring access from remote JMX clients. Ignored if jmx-manager is false. The Default RMI port is 1099.	1099
jmx-manager-ssl	If true and <code>jmx-manager-port</code> is not zero, the JMX Manager accepts only SSL connections. The ssl-enabled property does not apply to the JMX Manager, but the other SSL properties do. This allows SSL to be configured for just the JMX Manager without needing to configure it for the other GemFire connections. Ignored if <code>jmx-manager</code> is false.	false
jmx-manager-start	If true, this member starts a JMX Manager when it creates a cache. In most cases you should not set this property to true because a JMX Manager is automatically started when needed on a member that sets <code>jmx-manager</code> to true. Ignored if jmx-manager is false.	false
jmx-manager-update-rate	The rate, in milliseconds, at which this member pushes updates to any JMX Managers. Currently this value should be greater than or equal to the <code>statistic-sample-rate</code> . Setting this value too high causes gfsh and GemFire Pulse to see stale values.	2000

## Stopping a JMX Manager

To stop a JMX Manager using gfsh, simply shut down the locator or server hosting the JMX Manager.

For a locator:

```
gfsh>stop locator --name=locator1
Stopping Locator running in C:\vFabricGemFire70\Latest\locator1 on
GemFireStymon
[10334] as locator1...
Process ID: 3936
Log File: C:\vFabricGemFire70\Latest\locator1\locator1.log
.....
No longer connected to GemFireStymon[1099].
```

For a server:

```
gfsh>stop server --name=server1
Stopping Cache Server running in C:\vFabricGemFire70\Latest\server1 on
GemFireStymon[40404] as server1...
Process ID: 1156
Log File: C:\vFabricGemFire70\Latest\server1\server1.log
.....
No longer connected to GemFireStymon[1099].
```

Notice that gfsh has automatically disconnected you from the stopped JMX Manager.

To stop a JMX manager using the management API, use the `ManagementService stopManager()` method to stop a member from being a JMX Manager.

When a Manager stops, it removes all federated MBeans from other members from its Platform MBeanServer. It also emits a notification to inform other members that it is no longer considered a JMX Manager.

## Federated MBean Architecture

vFabric GemFire uses MBeans to manage and monitor different parts of GemFire. GemFire's federated MBean architecture is scalable and allows you to have a single-agent view of a GemFire distributed system.

### Federation of GemFire MBeans and MBeanServers

Federation of the MBeanServers means that one member, the JMX Manager Node, can provide a proxied view of all the MBeans that the MBeanServer hosts. Federation also means that operations and notifications are spread across the distributed system.

GemFire federation takes care of the following functionality:

- MBean proxy creation
- MBean state propagation
- Notifications propagation
- Operation invocation

### MBean Proxy Naming Conventions

Each GemFire MBean follows a particular naming convention for easier grouping. For example:

```
GemFire:type=Member,service=LockService,name=<dlsName>,memberName=<memberName>
```

At the JMX Manager node, this MBean will be registered with GemFire/<memberId> as domain.

The following are some sample MBean names:

MemberMBean:

```
GemFire:type=Member,member=<Node1>
```

## Use of MXBeans

In its Management API, vFabric GemFire provides MXBeans to ensure that any MBeans that are created are usable by any client, including remote clients, without requiring the client to access specific classes in order to access contents of the MBean.

### MBean Proxy Creation

GemFire proxies are inherently local MBeans. Every GemFire JMX manager member hosts proxies pointing to the local MBeans of every managed node. Proxy MBeans will also emit any notification emitted by local MBeans in managed nodes when an event occurs in that managed node.

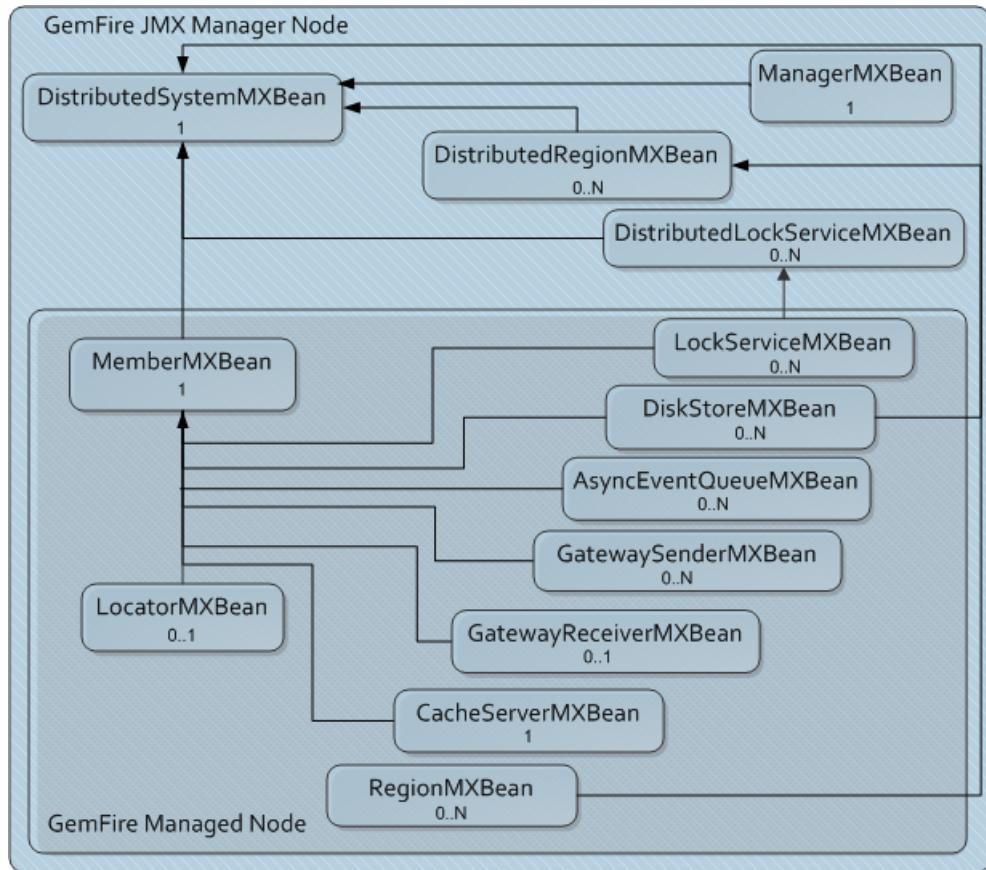


**Note:** Aggregate MBeans on the JMX Manager node are not proxied.

### List of GemFire JMX MBeans

This topic provides descriptions for the various management and monitoring MBeans that are available in GemFire.

The following diagram illustrates the relationship between the different JMX MBeans that have been developed to manage and monitor vFabric GemFire.



## JMX Manager MBeans

This section describes the MBeans that are available on the JMX Manager node.

The JMX Manager node includes all local beans listed under [Managed Node MBeans](#) on page 440 and the following beans that are available only on the JMX Manager node:

- [ManagerMXBean](#) on page 438
  - [DistributedSystemMXBean](#) on page 439
  - [DistributedRegionMXBean](#) on page 439
  - [DistributedLockServiceMXBean](#) on page 439

## ManagerMXBean

Represents the GemFire Management layer for the hosting member. Controls the scope of management. This MBean provides `start` and `stop` methods to turn a managed node into a JMX Manager node or to stop a node from being a JMX Manager. For potential managers (`jmx-manager=true` and `jmx-manager-start=false`), this MBean is created when a Locator requests it.



**Note:** You must configure the node to allow it to become a JMX Manager. See [Configuring a JMX Manager](#) on page 435 for configuration information.

MBean Details	
Scope	ALL
Proxied	No

Object Name	GemFire:type=Member, service=Manager,member=<name-or-dist-member-id>
Instances Per Node	1

See the [com.gemstone.gemfire.management.ManagerMXBean](#) JavaDocs for information on available MBean methods and attributes.

### DistributedSystemMXBean

System-wide aggregate MBean that provides a high-level view of the entire distributed system including all members (cache servers, peers, locators) and their caches. At any given point of time, it can provide a snapshot of the complete distributed system and its operations.

The DistributedSystemMXBean provides APIs for performing distributed system-wide operations such as backing up all members, shutting down all members or showing various distributed system metrics.

You can attach a standard JMX NotificationListener to this MBean to listen for notifications throughout the distributed system. See [GemFire JMX MBean Notifications](#) on page 444 for more information.

This MBean also provides some MBean model navigation APIS. These APIs should be used to navigate through all the MBeans exposed by a GemFire System.

MBean Details	
Scope	Aggregate
Proxied	No
Object Name	GemFire:type=Distributed,service=System
Instances Per Node	1

See the [com.gemstone.gemfire.management.DistributedSystemMXBean](#) JavaDocs for information on available MBean methods and attributes.

### DistributedRegionMXBean

System-wide aggregate MBean of a named region. It provides a high-level view of a region for all members hosting and/or using that region. For example, you can obtain a list of all members that are hosting the region. Some methods are only available for partitioned regions.

MBean Details	
Scope	Aggregate
Proxied	No
Object Name	GemFire:type=Distributed,service=Region,name=<regionName>
Instances Per Node	0..N

See the [com.gemstone.gemfire.management.DistributedRegionMXBean](#) JavaDocs for information on available MBean methods and attributes.

### DistributedLockServiceMXBean

Represents a named instance of DistributedLockService . Any number of DistributedLockService can be created in a member.

A named instance of DistributedLockService defines a space for locking arbitrary names across the distributed system defined by a specified distribution manager. Any number of DistributedLockService instances can be

created with different service names. For all processes in the distributed system that have created an instance of DistributedLockService with the same name, no more than one thread is permitted to own the lock on a given name in that instance at any point in time. Additionally, a thread can lock the entire service, preventing any other threads in the system from locking the service or any names in the service.

MBean Details	
Scope	Aggregate
Proxied	No
Object Name	GemFire:type=Distributed,service=Region,name=<regionName>
Instances Per Node	0..N

See the [com.gemstone.gemfire.management.DistributedLockServiceMXBean JavaDocs](#) for information on available MBean methods and attributes.

### Managed Node MBeans

This section describes the MBeans that are available on all managed nodes.

MBeans that are available on all managed nodes include:

- [MemberMXBean](#) on page 440
- [CacheServerMXBean](#) on page 441
- [RegionMXBean](#) on page 441
- [LockServiceMXBean](#) on page 441
- [DiskStoreMXBean](#) on page 441
- [GatewayReceiverMXBean](#) on page 442
- [GatewaySenderMXBean](#) on page 442
- [AsyncEventQueueMXBean](#) on page 442
- [LocatorMXBean](#) on page 443

JMX Manager nodes will have managed node MBeans for themselves since they are also manageable entities in the distributed system.

### MemberMXBean

Member's local view of its connection and cache. It is the primary gateway to manage a particular member. It exposes member level attributes and statistics. Some operations like `createCacheServer()` and `createManager()` will help to create some GemFire resources. Any JMX client can connect to the MBean server and start managing a GemFire Member by using this MBean.

See [MemberMXBean Notifications](#) on page 445 for a list of notifications emitted by this MBean.

MBean Details	
Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,member=<name-or-dist-member-id>
Instances Per Node	1

See the [com.gemstone.gemfire.management.MemberMXBean JavaDocs](#) for information on available MBean methods and attributes.

## **CacheServerMXBean**

Represents the GemFire CacheServer. Provides data and notifications about server, subscriptions, durable queues and indices.

See [CacheServerMXBean Notifications](#) on page 445 for a list of notifications emitted by this MBean.

<b>MBean Details</b>	
Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=CacheServer,member=<name-or-dist-member-id>
Instances Per Node	1

See the `com.gemstone.gemfire.management.CacheServerMXBean` JavaDocs for information on available MBean methods and attributes.

## **RegionMXBean**

Member's local view of region.

<b>MBean Details</b>	
Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=Region,name=<regionName>,member=<name-or-dist-member-id>
Instances Per Node	0..N

See the `com.gemstone.gemfire.management.RegionMXBean` JavaDocs for information on available MBean methods and attributes.

## **LockServiceMXBean**

Represents a named instance of a LockService . Any number of LockServices can be created in a member.

<b>MBean Details</b>	
Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=LockService,name=<dlsName>,member=<name-or-dist-member-id>
Instances Per Node	0..N

See the `com.gemstone.gemfire.management.LockServiceMXBean` JavaDocs for information on available MBean methods and attributes.

## **DiskStoreMXBean**

Represents a DiskStore object which provides disk storage for one or more regions

<b>MBean Details</b>	
Scope	Local

Proxied	Yes
Object Name	GemFire:type=Member,service=DiskStore,name=<name>,member=<name-or-dist-member-id>
Instances Per Node	0..N

See the `com.gemstone.gemfire.management.DiskStoreMXBean` JavaDocs for information on available MBean methods and attributes.

### GatewayReceiverMXBean

A GatewayReceiverMXBean represents a GatewayReceiver which is a local server for a remote WAN (or distributed system).

MBean Details	
Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=GatewayReceiver,gatewayReceiver=<gatewayReceiver>,member=<name-or-dist-member-id>
Instances Per Node	0..N

See the `com.gemstone.gemfire.management.GatewayReceiverMXBean` JavaDocs for information on available MBean methods and attributes.

### GatewaySenderMXBean

A GatewaySenderMXBean represents a GatewaySender which is a local proxy for a remote WAN (or distributed system).

MBean Details	
Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=GatewaySender,gateSender=<gateSender>,member=<name-or-dist-member-id>
Instances Per Node	0..N

### AsyncEventQueueMXBean

An AsyncEventQueueMXBean provides access to an AsyncEventQueue, which represent the channel over which events are delivered to the AsyncEventListener.

MBean Details	
Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=AsyncEventQueue=<AsyncEventQueue>,member=<name-or-dist-member-id>

Instances Per Node

0..N

See the [com.gemstone.gemfire.management.AsyncEventQueueMXBean JavaDocs](#) for information on available MBean methods and attributes.

## LocatorMXBean

A LocatorMXBean represents a locator .

MBean Details	
Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=Locator,port=<port>,member=<name-or-dist-member-id>
Instances Per Node	0..1

See the [com.gemstone.gemfire.management.LocatorMXBean JavaDocs](#) for information on available MBean methods and attributes.

## Browsing GemFire MBeans through JConsole

You can browse all the GemFire MBeans in your distributed system by using JConsole.

To view GemFire MBeans through JConsole, perform the following steps:

1. Start a gfsh prompt.
2. Connect to a running distributed system by either connecting to a locator with an embedded JMX Manager or connect directly to a JMX Manager. For example:

```
gfsh>connect --locator=GemFireStymon[10334]
```

or

```
gfsh>connect --jmx-manager=GemFireStymon[1099]
```

3. Start JConsole:

```
gfsh>start jconsole
```

If successful, the message Running JDK JConsole appears. The JConsole application launches and connects directly to the JMX Manager using RMI.

4. On the JConsole screen, click on the MBeans tab. Expand GemFire. Then expand each MBean to browse individual MBean attributes, operations and notifications.

The following is an example screenshot of the MBean hierarchy in a GemFire distributed system:

## GemFire JMX MBean Notifications

GemFire MBeans emit notifications when specific events occur or if an alert is raised in the GemFire system. Using standard JMX APIs, users can add notification handlers to listen for these events.

GemFire MBeans emit notifications when specific events occur or if an alert is raised in the system. Using standard JMX APIs, users can add notification handlers to listen for these events.

### Notification Federation

All notifications emitted from managed nodes are federated to all JMX Managers in the system.

These notifications are federated and then emitted by the `DistributedSystemMXBean`. If you attach a `javax.management.NotificationListener` to your `DistributedSystemMXBean`, the `NotificationListener` can listen to notifications from all `MemberMXBeans` and all `CacheServerMXBeans`.

### Attaching Listeners to MXBeans

When you attach a notification listener to the `DistributedSystemMXBean`, the `DistributedSystemMXBean` then acts as the notification hub for the entire distributed system. You do not have to attach a listener to each individual member or cache server MBean in order to listen to all the notifications in the distributed system.

The following is an example of attaching a `NotificationListener` to an MBean using the JMX MBeanServer API:

```
NotificationListener myListener = ...
ObjectName mbeanName = ...
MBeanServer.addNotificationListener(mbeanName, myListener, null, null);
```

JMX Managers will emit notifications for all distributed system members with two exceptions:

- If you use `cache.xml` to define resources such as regions and disks, then notifications for these resources are not federated to the JMX Manager. In those cases, the `DistributedSystemMXBean` cannot emit those notifications.
- If a JMX Manager is started after a resource has been created, the JMX Manager cannot emit notifications for that resource.

### System Alert Notifications

System alerts are GemFire alerts wrapped within a JMX notification. The JMX Manager registers itself as an alert listener with each member of the system, and by default, it receives all messages logged with the `SEVERE` alert level by any node in the distributed system. Consequently, the `DistributedSystemMXBean` will then emit notifications for these alerts on behalf of the `DistributedSystem`.

By default, the JMX Manager registers itself to send notifications only for `SEVERE` level alerts. To change the alert level that the JMX Manager will send notifications for, use the `DistributedMXBean.changeAlertLevel` method. Possible alert levels to set are `WARNING`, `ERROR`, `SEVERE`, and `NONE`. After changing the level, the JMX Manager will only emit that level of log message as notifications.

Notification objects include **type**, **source** and **message** attributes. System alerts also include the **userData** attribute. For system alerts, the notification object attributes correspond to the following:

- **type**: `system.alert`
- **source**: Distributed System ID
- **message**: alert message
- **userData**: name or ID of the member that raised the alert

### List of GemFire JMX MBean Notifications

This topic lists all available JMX notifications emitted by GemFire MBeans.

Notifications are emitted by the following MBeans:

### *MemberMXBean Notifications*

Notification Type	Notification Source	Message
gemfire.distributedsystem.cache.region.created	Member name or ID	Region Created with Name <Region Name>
gemfire.distributedsystem.cache.region.closed	Member name or ID	Region Destroyed/Closed with Name <Region Name>
gemfire.distributedsystem.cache.disk.created	Member name or ID	DiskStore Created with Name <DiskStore Name>
gemfire.distributedsystem.cache.disk.closed	Member name or ID	DiskStore Destroyed/Closed with Name <DiskStore Name>
gemfire.distributedsystem.cache.lockservice.created	Member name or ID	LockService Created with Name <LockService Name>
gemfire.distributedsystem.cache.lockservice.closed	Member name or ID	Lockservice Closed with Name <LockService Name>
gemfire.distributedsystem.gateway.sender.created	Member name or ID	GatewaySender Created in the VM
gemfire.distributedsystem.gateway.sender.started	Member name or ID	GatewaySender Started in the VM <Sender Id>
gemfire.distributedsystem.gateway.sender.stopped	Member name or ID	GatewaySender Stopped in the VM <Sender Id>
gemfire.distributedsystem.gateway.sender.paused	Member name or ID	GatewaySender Paused in the VM <Sender Id>
gemfire.distributedsystem.gateway.sender.resumed	Member name or ID	GatewaySender Resumed in the VM <Sender Id>
gemfire.distributedsystem.async.event.queue.created	Member name or ID	Async Event Queue is Created in the VM
gemfire.distributedsystem.gateway.receiver.created	Member name or ID	GatewayReceiver Created in the VM
gemfire.distributedsystem.gateway.receiver.started	Member name or ID	GatewayReceiver Started in the VM
gemfire.distributedsystem.gateway.receiver.stopped	Member name or ID	GatewayReceiver Stopped in the VM
gemfire.distributedsystem.cache.server.started	Member name or ID	Cache Server is Started in the VM
gemfire.distributedsystem.cache.server.stopped	Member name or ID	Cache Server is stopped in the VM
gemfire.distributedsystem.locator.started	Member name or ID	Locator is Started in the VM

### *CacheServerMXBean Notifications*

Notification Type	Notification Source	Message
gemfire.distributedsystem.cacheserver.client.joined	CacheServer MBean Name	Client joined with Id <Client ID>
gemfire.distributedsystem.cacheserver.client.left	CacheServer MBean Name	Client crashed with Id <Client ID>
gemfire.distributedsystem.cacheserver.client.crashed	CacheServer MBean name	Client left with Id <Client ID>

### *DistributedSystemMXBean Notifications*

Notification Type	Notification Source	Message
gemfire.distributedsystem.cache.member.joined	Name or ID of member who joined	Member Joined <Member Name or ID>
gemfire.distributedsystem.cache.member.departed	Name or ID of member who departed	Member Departed <Member Name or ID> has crashed = <true/false>
gemfire.distributedsystem.cache.member.suspect	Name or ID of member who is suspected	Member Suspected <Member Name or ID> By <Who Suspected>
system.alert.*	DistributedSystem("<DistributedSystem ID">")	Alert Message

## Configuring RMI Registry Ports and RMI Connectors

GemFire programmatically emulates out-of-box Java 1.6 JMX and creates a JMXServiceURL with RMI Registry and RMI Connector ports on all manageable members.

### Configuring JMX Manager Port and Bind Addresses

You can configure a specific connection port and address when launching a process that will host the GemFire JMX Manager by specifying values for the `jmx-manager-bind-address` and `jmx-manager-port`, which defines the RMI connection port.

The default GemFire JMX Manager RMI port is 1099. You may need to modify this default if 1099 is reserved for other uses.

### Using Out-of-the-Box RMI Connectors

If for some reason you need to use standard JMX RMI in your deployment for other monitoring purposes, set the GemFire property `jmx-manager-port` to 0 on any members where you want to use standard JMX RMI.

If you use out-of-the-box JMX RMI instead of starting an embedded GemFire JMX Manager, you should consider setting `-Dsun.rmi.dgc.server.gcInterval=Long.MAX_VALUE-1` when starting the JVM for customer applications and client processes. Every GemFire process internally sets this setting before creating and starting the JMX RMI connector in order to prevent full garbage collection from pausing processes.

## Executing gfsh Commands through the Management API

You can also use management APIs to execute `gfsh` commands programmatically.



**Note:** If you start a JMX manager programmatically and wish to enable command processing, you must add the absolute path of `gfsh-dependencies.jar` (located in `$GEMFIRE/lib` of your GemFire installation) to the `CLASSPATH` of your application. Do not copy this library to your `CLASSPATH` because this library refers to other dependencies in `$GEMFIRE/lib` by a relative path.

The following code samples demonstrate how to process and execute `gfsh` commands using the Java API.

First, retrieve a `CommandService` instance.



**Note:** The `CommandService` API is currently only available on JMX Manager nodes.

For example:

```
// Get existing CommandService instance or create new if it doesn't exist
commandService = CommandService.createLocalCommandService(cache);

// OR simply get CommandService instance if it exists, don't create new one
CommandService commandService =
CommandService.getUsableLocalCommandService();
```

Next, process the command and its output:

```
// Process the user specified command String
Result regionListResult = commandService.processCommand("list regions");

// Iterate through Command Result in String form line by line
while (regionListResult.hasNextLine()) {
    System.out.println(regionListResult.nextLine());
```

```
}
```

Alternatively, instead of processing the command, you can create a CommandStatement Object from the command string which can be re-used.

```
// Create a command statement that can be reused multiple times
CommandStatement showDeadLocksCmdStmt = commandService.createCommandStatement
("show dead-locks --file=deadlock-info.txt");
Result showDeadlocksResult = showDeadLocksCmdStmt.process();

// If there is a file as a part of Command Result, it can be saved to a
// specified directory
if (showDeadlocksResult.hasIncomingFiles()) {
    showDeadlocksResult.saveIncomingFiles(System.getProperty("user.dir")
+ "/commandresults");
}
```

## Management and Monitoring Programming Examples

One example demonstrates the use of an MBean server to manage and monitor a node in a distributed system, and the other example acts as the managed node.

These code samples are also provided in the `SampleCode/quickstart` directory of your GemFire product installation. For information on how to run the examples, see the QuickStart topic [Management and Monitoring Example](#) on page 85.

### JMX Manager Node Example

```
package quickstart;

import java.util.Set;
import javax.management.ObjectName;
import com.gemstone.gemfire.cache.Cache;
import com.gemstone.gemfire.cache.CacheFactory;
import com.gemstone.gemfire.distributed.DistributedMember;
import com.gemstone.gemfire.management.ManagementService;
import com.gemstone.gemfire.management.RegionMXBean;

/**
 * Example of a JMX Manager Node. It first creates a cache.
 * Then for each member of Distributed System it retrieves
RegionMXBean and uses its data.
 *
 * @author GemStone Systems, Inc.
 *
 * @since 7.0
 */
public class ManagerNode {

    /**
     * @param args
     */
    public static final String EXAMPLE_REGION_NAME =
"exampleRegion";

    public static void main(String[] args) throws Exception {
```

```

//Set waiting period for federation in milliseconds
final int JMX_WAIT_PERIOD_FOR_FEDERATION_UPDATE = 2500;

System.out
    .println("Connecting to the distributed system and
creating the cache.");

// Create the cache which causes the cache-xml-file to
be parsed
Cache cache = new CacheFactory()
    .set("name", "ManagerNode")
    .set("statistic-sampling-enabled", "true")
    .set("jmx-manager-start", "true")
    .set("jmx-manager", "true")
    .set("jmx-manager-port", "2576")
    .set("cache-xml-file",
"xml/Managed-node.xml").create();

// Retrieve service
System.out.println("Retrieving service ...");

//ManagementService.getManagementService() will create a
service in case not already created
//ManagementService.getExistingManagementService() will
return existing one, otherwise null.
ManagementService service =
ManagementService.getManagementService(cache);
System.out.println("Retrieved service ");

//Retrieve distributed system members
System.out.println("Retrieving distributed members ...");

Set<DistributedMember> dsMembers =
cache.getDistributedSystem()
    .getAllOtherMembers();
System.out.println("Retrieved " +dsMembers.size()+
" distributed members ");

//Sleep is needed for federation data to get published
System.out.println("Retrieving RegionMXBean for members
in DS ...");

try {
    Thread.sleep(JMX_WAIT_PERIOD_FOR_FEDERATION_UPDATE);
} catch (Exception e) {

}

for (DistributedMember dsMember : dsMembers) {
    System.out.println("Retrieved RegionMXBean for member
" + dsMember.getId());

    //Get region member bean name using region path.
    ObjectName regionMBeanName =
service.getRegionMBeanName(dsMember, "/" +
        + EXAMPLE_REGION_NAME);
    System.out.println("regionMBeanName " +

```

```

regionMBeanName);

        //Get Region MBean Proxy
        RegionMXBean regionMXBean =
service.getMBeanInstance(regionMBeanName,
                    RegionMXBean.class);

        // Validate regionMXbean
        if(regionMXBean != null){
            System.out.println("Entry count in " +
EXAMPLE_REGION_NAME + " is =" + regionMXBean.getEntryCount());

            }else{
                System.out.println("Retrieved RegionMXBean is null.");

            }
        }

        // Close the cache and disconnect from GemFire distributed
        system
        System.out.println("Closing the cache and
disconnecting.");
        cache.close();

        //Complete Managed Node by pressing enter in managed node
        console
        System.out.println(" ");
        System.out.println("Press enter in Managed Node");

    }
}

```

### GemFire Managed Node Example

```

package quickstart;

import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.gemstone.gemfire.cache.Cache;
import com.gemstone.gemfire.cache.CacheFactory;
import com.gemstone.gemfire.cache.Region;
import com.gemstone.gemfire.management.ManagementService;
import com.gemstone.gemfire.management.RegionMXBean;

/**
 * In this example of Managed Node.
 * 1. It creates a cache and adds entries in it.
 * 2. From ManagementService it retrieves service.
 * 3. From service gets Region Mbean.
 *
 *
 * @author GemStone Systems, Inc.
 *
 * @since 7.0
 */

public class ManagedNode {

```

```

    /**
     * @param args
     */
    public static final String EXAMPLE_REGION_NAME =
"exampleRegion";

    public static void main(String[] args) throws Exception {

        //Set waiting period for federation in milliseconds
        final int JMX_WAIT_PERIOD_FOR_FEDERATION_UPDATE = 2500;

        System.out
            .println("Connecting to the distributed system and
creating the cache.");

        // Create the cache which causes the cache-xml-file to
        // be parsed
        System.out.println("Creating cache using Mbean.xml");
        Cache cache = new CacheFactory().set("name",
"ManagedNode")
            .set("log-level", "fine")
            .set("statistic-sampling-enabled",
"true")
            .set("cache-xml-file",
"xml/Managed-node.xml").create();

        System.out.println("Created cache using Mbean.xml");

        // Create one sample region
        Region<Object, Object> region = cache.getRegion( "/" +
EXAMPLE_REGION_NAME);

        // Add some entries to region
        if (region != null) {
            System.out.println("Adding key and value to region in
member "
+
cache.getDistributedSystem().getDistributedMember().getId());

            for (int count = 0; count < 10; count++) {
                String key = "key" + count;
                String value = "value" + count;
                region.put(key, value);
            }
        }

        System.out.println("Retrieving service using cache ...");

        // Retrieve service in this node
        ManagementService service = ManagementService
            .getExistingManagementService(cache);
        System.out.println("Retrieved service.");

        System.out.println("Retrieving RegionMXBean for " +
EXAMPLE_REGION_NAME);

        try{
            Thread.sleep(JMX_WAIT_PERIOD_FOR_FEDERATION_UPDATE);
        }
    }
}

```

```
        }catch(Exception e){  
    }  
  
    RegionMXBean regionBean = service.getLocalRegionMBean(" / "  
        + EXAMPLE_REGION_NAME);  
  
    if (regionBean != null) {  
        System.out.println("Retrieved RegionMXBean for " +  
EXAMPLE_REGION_NAME);  
        System.out.println("Entry count for " +  
EXAMPLE_REGION_NAME + " is="  
            + regionBean.getEntryCount());  
  
        System.out.println("Start Manager Node and wait till  
it completes");  
    } else {  
        System.out.println("Could not retriev RegionMXBean for  
"  
            + EXAMPLE_REGION_NAME);  
    }  
  
    // Wait till user input, after manager node completes  
    BufferedReader bufferedReader = new BufferedReader(new  
InputStreamReader(  
        System.in));  
    bufferedReader.readLine();  
  
    // Close the cache and disconnect  
    System.out.println("Closing the cache and  
disconnecting.");  
    cache.close();  
}  
}
```



# Chapter 40

---

## Heap Use and Management

The vFabric GemFire resource manager works with your JVM's tenured garbage collection (GC) to control heap use and protect your member from hangs and crashes due to memory overload.

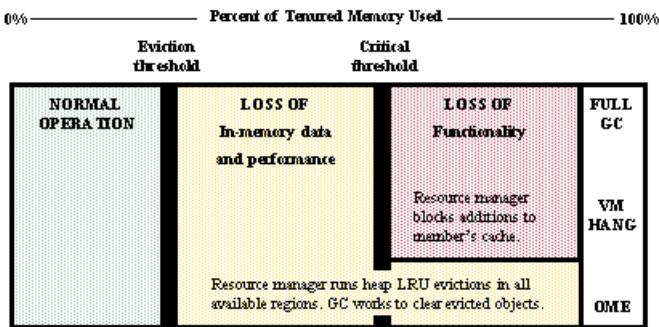
### How the Resource Manager Works

The resource manager prevents the cache from consuming too much memory by evicting old data. If the garbage collector is unable to keep up, the resource manager refuses additions to the cache until the collector has freed an adequate amount of memory.

You can use the resource manager in any vFabric GemFire member, but you may not want to use it everywhere. For some members it might be better to occasionally restart after a hang or OME crash than to evict data and/or refuse distributed caching activities. Also, members that do not risk running past their memory limits would not benefit from the overhead required to run the resource manager. Cache servers are often configured to use the manager because they generally host more data and have more data activity than other members, requiring greater responsiveness in data cleanup and collection.

The resource manager has two threshold settings, each expressed as a percentage of the total tenured heap. Both are disabled by default.

1. **Eviction Threshold.** Above this, the manager orders evictions for all regions with eviction-attributes set to `lru-heap-percentage`. This prompts dedicated background evictions, independent of any application threads and it also tells all application threads adding data to the regions to evict at least as much data as they add. The JVM garbage collector removes the evicted data, reducing heap use. The evictions continue until the manager determines that heap use is again below the eviction threshold.
2. **Critical Threshold.** Above this, all activity that might add data to the cache is refused. This threshold is set above the eviction threshold and is intended to allow the eviction and GC work to catch up. This JVM, all other JVMs in the distributed system, and all clients to the system receive `LowMemoryException` for operations that would add to this critical member's heap consumption. Activities that fetch or reduce data are allowed. For a list of refused operations, see the Javadocs for the `ResourceManager` method `setCriticalHeapPercentage`.



When heap use passes the eviction threshold, in either direction, the manager logs an info-level message. When it passes the critical threshold, the manager logs an error-level message.



**Note:** Avoid passing the critical threshold. It might be better than a hang or an OME, but the GemFire member that becomes a critical member is a read-only member that refuses cache updates for all of its regions, including incoming distributed updates.

For more information, see `com.gemstone.gemfire.cache.control.ResourceManager` in the online GemFire API documentation.

## How Background Eviction Is Performed

When the manager kicks off evictions:

1. From all regions in the local cache that are configured for heap LRU eviction, the background eviction manager creates a randomized list containing one entry for each partitioned region bucket (primary or secondary) and one entry for each non-partitioned region. So each partitioned region bucket is treated the same as a single, non-partitioned region.
2. The background eviction manager starts four evictor threads for each processor on the local machine. The manager passes each thread its share of the bucket/region list. The manager divides the bucket/region list as evenly as possible by count, and not by memory consumption.
3. Each thread iterates round-robin over its bucket/region list, evicting one LRU entry per bucket/region until the resource manager sends a signal to stop evicting.

See also [Memory Requirements for Cached Data](#) on page 989.

## Control Heap Use with the Resource Manager

Resource manager behavior is closely tied to the triggering of GC activities, the use of concurrent GCs in the JVM, and the number of parallel GC threads used for concurrency.

The recommendations provided here for using the manager assume you have a solid understanding of your Java VM's heap management and garbage collection service.

For the members where you want to implement the resource manager functionality:

1. Configure GemFire for heap LRU management.
2. Set the JVM GC tuning parameters to handle heap and garbage collection in conjunction with the GemFire manager.
3. Monitor and tune heap LRU configurations and your GC configurations.
4. Before going into production, run your system tests with application behavior and data loads that approximate your target systems so you can tune as well as possible for production needs.
5. In production, keep monitoring and tuning to meet changing needs.

## Configure GemFire for Heap LRU Management

The configuration terms used here are `cache.xml` elements and attributes, but you can also configure through the `com.gemstone.gemfire.cache.control.ResourceManager` and Region APIs.

1. Set the resource-manager `critical-heap-percentage` threshold. This should be as close to 100 as possible while still low enough so the manager's response can prevent the member from hanging or getting `OutOfMemoryError`. The threshold is zero (no threshold) by default.



**Note:** When you set this threshold, it also enables a query monitoring feature that prevents most out-of-memory exceptions when executing queries or creating indexes. See [Monitoring Queries for Low Memory](#) on page 356

for more information.

2. Set the resource-manager `eviction-heap-percentage` threshold to a value lower than the critical threshold. This should be as high as possible while still low enough to prevent your member from reaching the critical threshold. The threshold is zero (no threshold) by default.
3. Decide which regions will participate in heap eviction and set their `eviction-attributes` to `lru-heap-percentage`. See [Eviction](#) on page 248. The regions you configure for eviction should have enough data activity for the evictions to be useful and should contain data your application can afford to delete or offload to disk.

Example:

```
<resource-manager critical-heap-percentage="80"
eviction-heap-percentage="60" />
<region refid="REPLICATE_HEAP_LRU" />
```

## Set the JVM's GC Tuning Parameters

See your JVM documentation for all JVM-specific settings that can be used to improve GC response.

At a minimum, do the following:

1. Set the initial and maximum heap switches, `-Xms` and `-Xmx`, to the same values.
2. Configure your JVM for concurrent mark-sweep collector garbage collection.
3. If your JVM allows, configure it to initiate concurrent mark-sweep collection when heap use is at least 10% lower than your setting for the resource manager `eviction-heap-percentage`. You want the collector to be working when GemFire is evicting or the evictions will not result in more free memory. For example if the `eviction-heap-percentage` is set to 65, set your garbage collection to start when the heap use is no higher than 55%.

JVM	Conc mark-sweep switch flag	CMS initiation (begin at heap % N)
Sun HotSpot	<code>-XX:+UseConcMarkSweepGC</code>	<code>-XX:CMSInitiatingOccupancyFraction=N</code>
JRockit	<code>-Xgc:gencon</code>	<code>-XX:gctTrigger:N</code>
IBM	<code>-XgcPolicy:gencon</code>	N/A

For the `gfsh start server` command, pass these settings with the `--J` switch, like  
`--J=-XX:+UseConcMarkSweepGC`.

## Monitor and Tune Heap LRU Configurations

In tuning the resource manager, your central focus should be keeping the member below the critical threshold. The critical threshold is provided to avoid member hangs and crashes, but because of its exception throwing behavior for distributed updates, the time spent in critical negatively impacts the entire distributed system. To

stay below critical, tune so that the GemFire eviction and the JVM's GC respond adequately when the eviction threshold is reached.

Use the statistics provided by your JVM to make sure your memory and GC settings are sufficient for your needs.

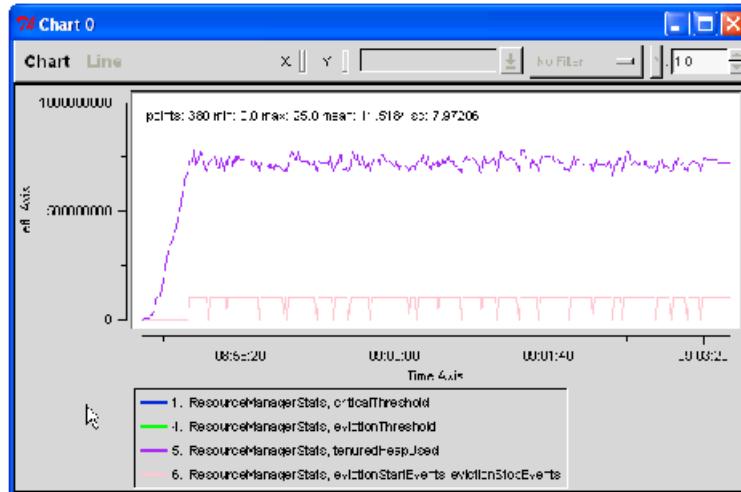
The GemFire ResourceManagerStats provide information about memory use and the manager thresholds and eviction activities.

If you are spiking above the critical threshold on a regular basis, try lowering the eviction threshold. If you never go near critical, you might raise the eviction threshold to gain more usable memory without the overhead of unneeded evictions or GC cycles.

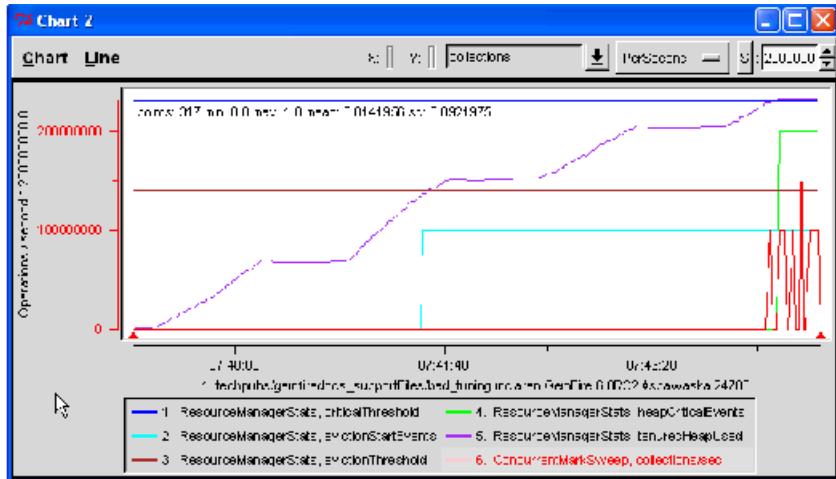
The settings that will work well for your system depend on a number of factors, including these:

- The size of the data objects you store in the cache. Very large data objects can be evicted and garbage collected relatively quickly. The same amount of space in use by many small objects takes more processing effort to clear and might require lower thresholds to allow eviction and GC activities to keep up.
- Application behavior. Applications that quickly put a lot of data into the cache can more easily overrun the eviction and GC capabilities. Applications that operate more slowly may be more easily offset by eviction and GC efforts, possibly allowing you to set your thresholds higher than in the more volatile system.
- Your choice of JVM. Each JVM has its own GC behavior, which affects how efficiently the collector can operate, how quickly it kicks in when needed, and other factors.

In this sample statistics chart in VSD, the manager's evictions and the JVM's GC efforts are good enough to keep heap use very close to the eviction threshold. The eviction threshold could be increased to a setting closer to the critical threshold, allowing the member to keep more data in tenured memory without the risk of overwhelming the JVM. This chart also shows the blocks of times when the manager was running cache evictions.



In this next chart, it looks like the manager's evictions are kicking in at the right time, but the concurrent mark sweep GC is not starting soon enough to keep memory use in check. It might be that it is not configured to start as soon as it should. It should be started just before the eviction threshold is reached. Or there might be some other issue with the garbage collection service.



## Resource Manager Example Configurations

These examples set the critical threshold to 85 percent of the tenured heap and the eviction threshold to 75 percent.

The region `bigDataStore` is configured to participate in the resource manager's eviction activities.

- XML:

```
<cache>
  <resource-manager critical-heap-percentage="85"
    eviction-heap-percentage="75" />
  <region name="bigDataStore" refid="PARTITION_HEAP_LRU" />
</cache>
```

- Java:

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

ResourceManager rm = cache.getResourceManager();
rm.setCriticalHeapPercentage(85);
rm.setEvictionHeapPercentage(75);

RegionFactory rf =
  cache.createRegionFactory(RegionShortcut.PARTITION_HEAP_LRU);
Region region = rf.create("bigDataStore");
```

## Use Case for the Example Code

This is one possible scenario for the configuration used in the examples:

- A 64-bit Sun Java VM 1.6 JVM, with 8 Gb of heap space on an 4 CPU system running Linux.
- The data region `bigDataStore` has approximately 2-3 million small values with average entry size of 512 bytes. So approximately 4-6 Gb of the heap is for region storage.
- The member hosting the region also runs an application that may take up to 1 Gb of the heap.
- The application must never run out of heap space and has been crafted such that data loss in the region is acceptable if the heap space becomes limited due to application issues, so the default `lru-heap-percentage` action `destroy` is suitable.
- The application's service guarantee makes it very intolerant of `OutOfMemoryExceptions`. Testing has shown that leaving 15% head room above the critical threshold when adding data to the region gives 99.5%

uptime with no OutOfMemoryExceptions, when configured with the CMS garbage collector using -XX:CMSInitiatingOccupancyFraction=70.

# Chapter 41

## Disk Storage

With vFabric GemFire disk stores, you can persist data to disk as a backup to your in-memory copy and overflow data to disk when memory use gets too high.

### How Disk Stores Work

The overflow feature uses disk stores as an extension of the in-memory cache. Persistence uses disk stores to store a redundant copy of the data.

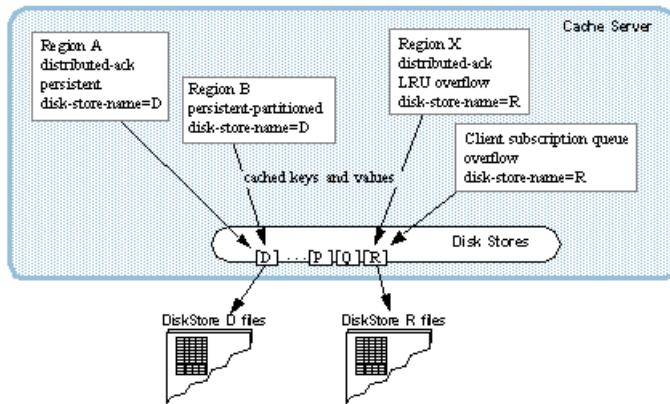
These two options can be used individually or together.

Disk storage is available for these cached data types:

- **Cached regions.** Persist and/or overflow data from your cached data regions.
- **Server's client subscription queues.** Overflow the messaging queues to control memory use.
- **Gateway sender queues.** Persist these for high availability. These queues always overflow.
- **PDX serialization metadata.** Persist metadata about objects you serialize using GemFire PDX serialization.

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.



## What GemFire Writes to the Disk Store

The following list describes the items that 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 cached regions, these are the names you give the regions. For all other data, these are GemFire internal region names.
- For each region:
  - Region configuration attributes pertaining to loading and capacity management, used to load the data quickly on startup.
  - Region data operations.

GemFire does not write indexes to disk.

## 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, 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 `gfsh` command-line interface. The `gfsh` command-line interface joins the distributed system and sends requests to members that have disk stores.
- 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 maintenance operations on the offline disk store, like file compaction and validation by using the `gfsh` command-line interface.



**Note:** 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 files, 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 Names

File names have three parts:

#### First Part of File Name: Usage Identifier

Values	Used for	Examples
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

Values	Used for	Examples
DRLK_IF	Access control - locking the disk store.	DRLK_IFOverflowDS1.lk, DRLK_IFDEFAULT.lk

### Second Part of File Name: Disk Store Name

Values	Used for	Examples
<disk store name>	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 of File Name: oplog Sequence Number

Values	Used for	Examples
Sequence number in the format _n	Oplog data files only. Numbering starts with 1.	OVERFLOWoverflowDS1_1.crf, BACKUPpersistDS1_2.crf, BACKUPpersistDS1_3.crf

### File Extensions

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-allocated 90% of the total max-oplog-size at creation.
drf	Oplog: delete operations	Pre-allocated 10% of the total max-oplog-size at creation.
krf	Oplog: key and crf offset information	Created after the oplog has reached the max-oplog-size. Used to improve performance at startup.

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 default disk store files for a 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 disk store's `max-oplog-size`, with the size divided between the `crf` and `drf` files. When the oplog is closed, vFabric GemFire shrinks the files to the space used in each file.

After the oplog is closed, GemFire also attempts to create a `krf` file, which contains the key names as well as the offset for the value within the `crf` file. Although this file is not required for startup, if it is available, it will improve startup performance by allowing GemFire to load the entry values in the background after the entry keys are loaded. See [How Startup and Shutdown Work with Disk Stores](#) on page 468.

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.



**Note:** 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. 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.

```
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 Disk Store Ologs 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.
```



**Note:** 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
```

## Configuring Disk Stores

In addition to the disk stores you specify, vFabric GemFire has a default disk store that it uses when disk use is configured with no disk store name specified. You can modify default disk store behavior.

### Designing and Configuring Disk Stores

You define disk stores in your cache, then you assign them to your regions and queues by setting the `disk-store-name` attribute in your region and queue configurations.



**Note:** Besides the disk stores you specify, vFabric GemFire has a default disk store that it uses when disk use is configured with no disk store name specified. By default, this disk store is saved to the application's working directory. You can change its behavior, as indicated in [Create and Configure Your Disk Stores](#) on page 464 and [Modifying the Default Disk Store](#) on page 467.

The following sections describe how to design and configure your disk stores:

- [Design Your Disk Stores](#) on page 463
- [Create and Configure Your Disk Stores](#) on page 464
- [Configuring Regions, Queues, and PDX Serialization to Use the Disk Stores](#) on page 465

### Design Your Disk Stores

Before you begin, you should understand GemFire [Basic Configuration and Programming](#) on page 131.

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.
  - 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 sender 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. GemFire creates each oplog file at the `max-oplog-size`, which defaults to 1 GB. Obsolete operations

are only removed from the oplogs during compaction, so you need enough space to store all operations that are done between compactions. For regions where you are doing a mix of updates and deletes, if you use automatic compaction, a good upper bound for the required disk space is

```
(1 / (1 - (compaction_threshold/100)) ) * data size
```

where data size is the total size of all the data you store in the disk store. So, for the default compaction-threshold of 50, the disk space is roughly twice your data size. Note that the compaction thread could lag behind other operations, causing disk use to rise above the threshold temporarily. If you disable automatic compaction, the amount of disk required depends on how many obsolete operations accumulate between manual compactions.

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.

## Create and Configure Your Disk Stores

1. 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.
2. 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.
3. Configure each disk store:
  - a. Set the name. If you are modifying the default disk store configuration, use "DEFAULT".

Example:

```
name="serverOverflow"
```

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

Example:

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

- c. As needed, modify the store's file compaction behavior. In conjunction with this, plan and program for any manual compaction.

Example:

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

- d. As needed, modify the maximum size of a single oplog. 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.

Example:

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

- e. As needed, modify queue management parameters for asynchronous queueing to the disk store. You can configure any region for synchronous or asynchronous queueing (region attribute `disk-synchronous`). Server queues and gateway sender queues always operate synchronously. When either the `queue-size` or `time-interval` is reached, enqueued data is flushed to disk. You can also synchronously flush unwritten data to disk through the `DiskStore flushToDisk` method.

Example:

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

- f. As needed, modify the size of the buffer used for writing to disk.

Example:

```
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>
```

When you start your system, all defined disk stores will be used by GemFire as configured.

#### Related Topics

[com.gemstone.gemfire.cache.DiskStore](#)

## Configuring Regions, Queues, and PDX Serialization to Use the Disk Stores

Example of using a named disk store for region persistence and overflow:

```
<region refid="PARTITION_PERSISTENT_OVERFLOW"
    disk-store-name="persistOverflow1" />
```

Example of using the default disk store for gateway sender queue persistence:

```
<cache>
    <gateway-sender id="persistedsender1" parallel="true"
        remote-distributed-system-id="1"
        is-persistent="true"
        disk-store-name="diskStoreA"
        maximum-queue-memory="100" />
    ...
</cache>
```

Example of using the default disk store for server subscription queue overflow:

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

```
        capacity="10000" disk-store-name="queueOverflow2"/>
</cache-server>
```

Example of using a named disk store for PDX serialization metadata:

```
<pdx read-serialized="true"
      persistent="true" disk-store-name="SerializationDiskStore">
</pdx>
```

#### Related Topics

[<region-attributes>](#) on page 882

[cache.xml Quick Reference](#) on page 861

[Data Serialization](#) on page 257

## Disk Store Configuration Parameters

You define your disk stores in `<disk-store>` subelements of your cache declaration in `cache.xml`. All disk stores are available for use by all of your regions and queues.

These `<disk-store>` attributes and subelements have corresponding setter and getter methods in the `com.gemstone.gemfire.cache.DiskStoreFactory` and `com.gemstone.gemfire.cache.DiskStore` APIs.

## Disk Store Configuration Attributes and Elements

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 <code>compaction-threshold</code> .	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.	1024
queue-size	For asynchronous queueing. The maximum number of operations to allow into the write queue before automatically flushing the queue. Operations that would add entries to	0

disk-store attribute	Description	Default
	the queue 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.	
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 with the following contents:

- 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.

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>
```



**Note:** 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.

### Modifying the Default Disk Store

You can modify the behavior of the default disk store by specifying the attributes you want for the disk store named "DEFAULT".

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

For example, these region and queue 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 of using the default disk store for region persistence and overflow:

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

Example of using the default disk store for gateway sender queue persistence and overflow:

```
<cache>
  <gateway-sender id="persistedsender1" parallel="true"
    remote-distributed-system-id="1"
    is-persistent="true"
    maximum-queue-memory="100" />
  ...
</cache>
```

Example of using the default disk store for server subscription queue overflow:

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

### **Change the Behavior of the Default Disk Store**

GemFire initializes the default disk store with the default disk store configuration settings. 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>
```

#### **Related Topics**

[com.gemstone.gemfire.cache.DiskStore](#)

## **Optimizing a System with Disk Stores**

Optimize availability and performance by following the guidelines in this section.

1. When you start your system, start all the members that have persistent regions at roughly the same time. Create and use startup scripts for consistency and completeness.
2. Shut down your system using the gfsh shutdown command. This is an ordered shutdown that positions your disk stores for a faster startup.
3. Decide on a file compaction policy and, if needed, develop procedures to monitor your files and execute regular compaction.
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.
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.

## **Starting and Shutting Down a System with Disk Stores**

This section describes what happens during startup and shutdown and provides procedures for those operations.

### **How Startup and Shutdown Work with Disk Stores**

When you shut down a member that is persisting data, the data remains in the disk store files. When the member starts up again, the data is reloaded and reinitializes the member's persistent regions.

## Shutdown: Most Recent Data from the Last Run

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 gfsh shutdown command, all online data stores are synchronized before shutting down so all hold the most recent data copy.
  - Otherwise, different members might host different most recent buckets.

## 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:

1. 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.

```
Region /people has potentially stale data.
It is waiting for another member to recover the latest data.
My persistent id:
```

```
DiskStore ID: 6893751ee74d4fbd-b4780d844e6d5ce7
Name: server1
Location: /10.118.32.49:/home/dsmith/server1/.
```

Members with potentially new data:

```
[ 
  DiskStore ID: 160d415538c44ab0-9f7d97bae0a2f8de
  Name: server2
  Location: /10.118.32.49:/home/dsmith/server2/.
```

]

Use the "gemfire list-missing-disk-stores" command to see all disk stores that are being waited on by other members.

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.

2. 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.

Each member's persistent regions load and go online as quickly as possible, not waiting unnecessarily for other members to complete. For performance reasons, several actions occur asynchronously:

- If both primary *and* secondary buckets are persisted, data will be available when the primary buckets are loaded without waiting for the secondary buckets to load; the secondary buckets will load asynchronously.

- Entry keys first get loaded from the key file if this file is available (see information about the krf file in [Disk Store File Names and Extensions](#) on page 460). Once all keys are loaded, GemFire loads the entry values asynchronously. If a value is requested before it is loaded, the value will immediately be fetched from the disk store.

## Example Startup Scenarios

The following lists scenarios for starting up a system with disk stores after a shutdown:

- Stop order for a replicate persistent region
  1. Member A (MA) exits first, leaving persisted data on disk for RegionP.
  2. MB continues to run operations on RegionP, which update its disk store and leave the disk store for MA in a stale condition.
  3. MB exits, leaving the most up-to-date data on disk for RegionP.
- Restart order Scenario 1
  1. MB is started first. GemFire recognizes MB as having the most recent disk data for RegionP and initializes it from disk.
  2. MA is started, recovers its data from disk, and updates it as needed from the data in MB.
- Restart order Scenario 2
  1. MA is started first. GemFire recognizes that MA does not have the most recent disk data and waits for MB to start before creating RegionP in MA.
  2. MB is started. GemFire recognizes MB as having the most recent disk data for RegionP and initializes it from disk.
  3. MA recovers its RegionP data from disk and updates it as needed from the data in MB.

## Starting a System with Disk Stores

Starting members with persisted data at the same time gives you the most efficient startup.

To start a system with disk stores:

1. **Start all members with persisted data first and at the same time.** Exactly how you do this depends on your members.

When you start members with persisted data in parallel, you get the most efficient system startup. While they are initializing their caches, the members determine which have the most recent region data, and initialize their caches from that.



**Note:** For replicates, where you can define persistence only in some of the region's host members, start the persistent replicates before the non-persistent replicates to make sure the data is recovered from disk.

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.

```
#!/bin/bash
ssh servera "cd /my/directory; gfsh server start --name=servera &
ssh serverb "cd /my/directory; gfsh server start --name=serverb &

STATUS=0;
for job in `jobs -p`;
do
echo $job
wait $job;
```

```
JOB_STATUS=$?;
test $STATUS -eq 0 && STATUS=$JOB_STATUS;
done
exit $STATUS;
```

2. **Respond to hangs on member startup.** 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 the gfsh show missing-disk-stores command. See [Handling Missing Disk Stores](#) on page 476.
  - a. If no disk stores are missing, your cache initialization is slow for some other reason. Check the information on member hangs in [Diagnosing System Problems](#) on page 606.
  - b. If disk stores are missing that you think should be there:
    - a. Make sure you have started the member. Check the logs for any failure messages. See [Logging](#) on page 543.
    - b. Make sure your disk store files are accessible. If you have moved your member or disk store files, you must update your disk store configuration to match.
  - c. If disk stores are missing that you know are lost, because you have deleted them or their files are otherwise unavailable, revoke them so the startup can continue. See [Handling Missing Disk Stores](#) on page 476.

## Shutting Down a System with Disk Stores

You use the gfsh command to shut down a system with disk stores.

To shut down the system:

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

```
shutdown
```

You must be connected to the distributed system in gfsh to execute this command.

The shutdown command 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 region data stores before shutdown. This means every disk store has the most recent data and does not require updates from other members at startup. See also [How Startup and Shutdown Work with Disk Stores](#) on page 468 and [shutdown](#) on page 712.

## Disk Store Management

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

See [Disk Store Commands](#) on page 660 for a list of available commands.

### Disk Store Management Commands and Operations

The gemfire command-line utility has been replaced with gfsh. For more information on gfsh commands, see [Using the GemFire SHell \(gfsh\)](#) on page 641 and [Disk Store Commands](#) on page 660.



**Note:** Each of these commands operates either on the online disk stores or offline disk stores, but not both.

gfsh Command	Online or Offline Command	See ...
alter disk-store	Off	<a href="#">Synchronizing Your Offline Disk Store with Your Cache on page 475</a>
compact disk-store	On	<a href="#">Running Compaction on Disk Store Log Files on page 473</a>
backup disk-store	On	<a href="#">Back Up and Restore a Disk Store on page 478</a>
compact offline-disk-store	Off	<a href="#">Running Compaction on Disk Store Log Files on page 473</a>
revoke missing-disk-store	On	<a href="#">Handling Missing Disk Stores on page 476</a>
show missing-disk-stores	On	<a href="#">Handling Missing Disk Stores on page 476</a>
shutdown	On	<a href="#">Shutting Down a System with Disk Stores on page 471</a>
validate offline disk-store	Off	<a href="#">Validating a Disk Store on page 472</a>

For complete command syntax of any gfsh command, run `help <command>` at the gfsh command line.

### Online Disk Store Operations

For online operations, gfsh must be connected to a distributed system via a JMX manager and sends the operation requests to the members that have disk stores. These commands will not run on offline disk stores.

### Offline Disk Store Operations

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

```
gfsh>compact offline-disk-store --name=mydiskstore --disk-dirs=MyDirs
```

Offline operations 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:

```
gfsh>compact offline-disk-store --name=DEFAULT --disk-dirs=s1
This disk store is in use by another process. "compact disk-store" can
be used to compact a disk store that is currently in use.
```

### Validating a Disk Store

The validate offline-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 command at these times:

- 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:

```
gfsh>validate offline-disk-store --name=ds1 --disk-dirs=hostB/bupDirectory
```

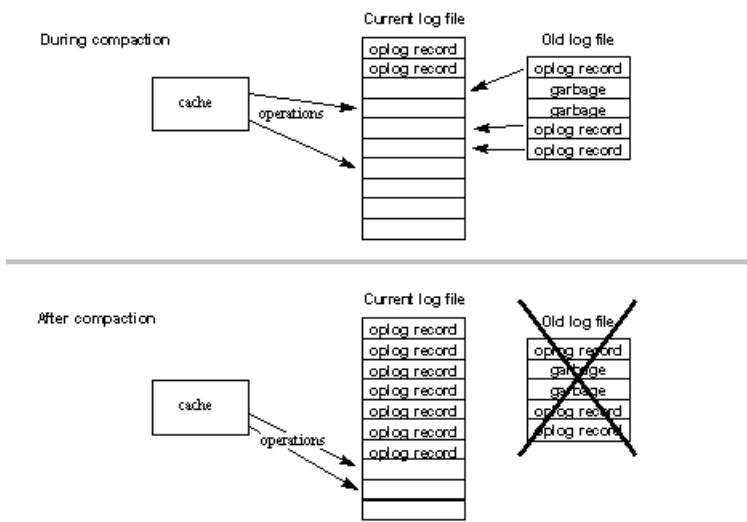
## Running Compaction on Disk Store Log Files

When a cache operation is added to a disk store, any preexisting operation record for the same entry becomes obsolete, and vFabric 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.

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.

## Log File Compaction for the Online Disk Store



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.

## Run 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 compaction.** When auto-compact is true, GemFire automatically compacts each oplog when its garbage content surpasses the compaction-threshold. This takes cycles from your other operations, so you may want to disable this and only do manual compaction, to control the timing.
- **Manual compaction.** To run manual compaction:

- Set the disk store attribute `allow-force-compaction` 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` are compacted into the current oplog.
- Run manual compaction as needed. GemFire has two types of manual compaction:
  - Compact the logs for a single online disk store through the API, with the `forceCompaction` method. This method first rolls the oplogs and then compacts them. Example:
 

```
myCache.getDiskStore("myDiskStore").forceCompaction();
```
  - Using `gfsh`, compact a disk store in a distributed system with the [compact disk-store](#) on page 677 command. Examples:
 

```
gfsh>compact disk-store --name=Disk1
```

```
gfsh>compact disk-store --name=Disk1 --group=MemberGroup1,MemberGroup2
```



**Note:** You need to be connected to a JMX Manager in `gfsh` to run this command.

## Run 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.

Using `gfsh`, compact individual offline disk stores with the [compact offline-disk-store](#) on page 677 command:

```
gfsh>compact offline-disk-store --name=Disk2 --disk-dirs=/Disks/Disk2
```

```
gfsh>compact offline-disk-store --name=Disk2 --disk-dirs=/Disks/Disk2
--max-oplog-size=512 -J=-Xmx1024m
```



**Note:** Do not perform offline compaction on the baseline directory of an incremental backup.

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 may need to increase your heap size with the `-J=-Xmx` parameter.

## Performance Benefits of 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 could run `compact disk-store` command 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-compactified 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

Reaching directory size limits during has different results depending on whether you are running an automatic or manual 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 Compaction Run

In this example offline compaction run listing, 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.

```
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$ gfsh
gfsh>validate offline-disk-store --name=ds1 --disk-dirs=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

gfsh>compact offline-disk-store --name=ds1 --disk-dirs=backupDirectory
Offline compaction removed 12 records.
Total number of region entries in this disk store is: 7

gfsh>exit

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$
```

### Synchronizing Your Offline Disk Store with Your Cache

You can take several actions to optimize disk store use and data loading at startup.

#### Change 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:

```
gfsh>alter disk-store --name=myDiskStoreName --region=/partitioned_region
--disk-dirs=/firstDiskStoreDir,/secondDiskStoreDir,/thirdDiskStoreDir
--initialCapacity=20
```

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

Example:

```
gfsh>alter disk-store --name=myDiskStoreName --region=/partitioned_region
--disk-dirs=/firstDiskStoreDir,/secondDiskStoreDir,/thirdDiskStoreDir
```

### **Take a Region Out of Your Cache Configuration and Disk Store**

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.



**Note:** 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:

```
gfsh>alter disk-store --name=myDiskStoreName --region=/partitioned_region
--disk-dirs=/firstDiskStoreDir,/secondDiskStoreDir,/thirdDiskStoreDir
--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.

#### **Show Missing Disk Stores**

Using gfsh, the show 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 stores, even when other data stores for the region are online, because their offline status may be causing PartitionOfflineExceptions in cache operations or preventing the system from satisfying redundancy.

Example:

Disk Store ID	Host	Directory
---------------	------	-----------

-----		-----
60399215-532b-406f-b81f-9b5bd8d1b55a		excalibur
/usr/local/gemfire/deploy/disk_store1		



**Note:** You need to be connected to JMX Manager in gfsh to run this command.



**Note:** 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. You can check on this by validating the disk store.
- Incorrect distributed system configuration for the member
- Network partitioning
- Drive failure

### Revoke Missing Disk Stores

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.



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

Use gfsh show missing-disk-stores to properly identify the disk store you need to revoke. The revoke command takes the disk store ID as input, as listed by that command.

Example:

```
gfsh>revoke missing-disk-store --id=60399215-532b-406f-b81f-9b5bd8d1b55a
Missing disk store successfully revoked
```

### Managing How Data is Written to Disk

You can configure GemFire to write immediately to disk and you may be able to modify your operating system behavior to perform buffer flushes more frequently.

Typically, GemFire writes disk data into the operating system's disk buffers and the operating system periodically flushes the buffers to disk. Increasing the frequency of writes to disk decreases the likelihood of data loss from application or machine crashes, but it impacts performance. Your other option, which may give you better performance, is to use GemFire's in-memory data backups. Do this by storing your data in multiple replicated regions or in partitioned regions that are configured with redundant copies. See [Region Types](#) on page 206.

## Modifying Disk Flushes for the Operating System

You may be able to change the operating system settings for periodic flushes. You may also be able to perform explicit disk flushes from your application code. For information on these options, see your operating system's documentation. For example, in Linux you can change the disk flush interval by modifying the setting `/proc/sys/vm/dirty_expire_centiseconds`. It defaults to 30 seconds. To alter this setting, see the Linux documentation for `dirty_expire_centiseconds`.

## Modifying GemFire to Flush Buffers on Disk Writes

You can have GemFire flush the disk buffers on every disk write. Do this by setting the system property `gemfire.syncWrites` to true at the command line when you start your GemFire member. You can only modify this setting when you start a member. When this is set, GemFire uses a Java `RandomAccessFile` with the flags "rwd", which causes every file update to be written synchronously to the storage device. This only guarantees your data if your disk stores are on a local device. See the Java documentation for `java.io.RandomAccessFile`.

To modify the setting for a GemFire application, add this to the java command line when you start the member:

```
-Dgemfire.syncWrites=true
```

To modify the setting for a cache server, use this syntax:

```
gfh>start server --name=... --J=-Dgemfire.syncWrites=true
```

## Back Up and Restore a Disk Store

You do backup and restore operations differently for online and offline distributed systems.

- [Online Backup](#) on page 478
- [What a Full Online Backup Saves](#) on page 480
- [What an Incremental Online Backup Saves](#) on page 481
- [Disk Store Backup Directory Structure and Contents](#) on page 481
- [Offline Members: Manual Catch-Up to an Online Backup](#) on page 481
- [Restore an Online Backup](#) on page 482
- [Offline File Backup and Restore](#) on page 482

### Online Backup

The gfsh backup operation creates a backup of disk stores for all members running in the distributed system when the backup command is invoked.



**Note:** Do not try to create backup files from a running system by using your operating system's file copy commands. You will get incomplete and unusable copies.

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.

### Preparing for Backup

1. You might want to compact your disk store before running the backup. If auto-compaction is turned off, you may want to do a manual compaction to save on how much data will be copied over your network by the backup. For more information on configuring a manual compaction, see [List item](#) on page 473
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 disk store files. VMware recommends that you back up:

- `cache.xml`
- `gemfire.properties`
- Application jar files
- Other files that the application needs when starting (a file that sets the classpath, for example)

Any directory that you specify is copied recursively, with any disk stores that are found excluded from this user-specified backup. Example:

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

4. Back up to a SAN (recommended) or 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 `gemfire.properties` file is required by the backup command so that it can connect to the specified distributed system and instruct members to back up their disk stores. Make sure that `locators` or `mcast-port` are correctly set in the `gemfire.properties` file to connect to the distributed system that you want to back up.
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

## Performing a Full Online Backup

1. If you have disabled auto-compaction, run manual compaction:

```
gfsh>compact disk-store --name=Disk1
```

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

```
gfsh>backup disk-store  
--dir=/export/fileServerDirectory/gemfireBackupLocation
```

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

```
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.

4. Validate the backup. To ensure that the backup can be recovered, validate the backed-up files. Run the validate offline-disk-store command on the backed-up files for each disk store.

```
cd 2010-04-10-11-35/straw_14871_53406_34322/diskstores/ds1
gemfire validate-disk-store ds1 dir0 dir1 [... dirN]
```

Repeat for all disk stores of all members.

### Performing an Incremental Backup

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

```
gfsh>backup disk-store
--dir=/export/fileServerDirectory/gemfireBackupLocation
--baselineDir=/export/fileServerDirectory/gemfireBackupLocation/2012-10-01-12-30
```

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

```
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 performing an incremental backup on all known members, you see a message like this:

```
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. The next time you perform a backup operation a full backup will be performed.

3. Repeat for all disk stores for all members.

### What a Full Online Backup Saves

For each member with persistent data, a full backup includes the following:

- Disk store files for all stores containing persistent region data.
- Files and 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>
```

- Deployed JAR files that you deployed using the gfsh [deploy](#) on page 682 command.
- Configuration files from the member startup.
  - gemfire.properties, including the properties with which the member was started.
  - cache.xml, if used.

These configuration 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

can override the files in the `.jar`. If you want to back up and restore these files, add them as custom `<backup>` elements.

- 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`.

## What an Incremental Online Backup Saves

An incremental backup saves the difference between the last backup and the current data. An incremental backup copies only operations logs that are not already present in the baseline directories for each member. For incremental backups, the restore script contains explicit references to operation logs in one or more previously chained incremental backups. When the restore script is run from an incremental backup, it also restores the operation logs from previous incremental backups that are part of the backup chain.

If members are missing from the baseline directory because they were offline or did not exist at the time of the baseline backup, those members place full backups of all their files into the incremental backup directory.

## Disk Store Backup Directory Structure and Contents

```
dsmith@dsmith-e6410: test_backup$ ls -R
./2012-10-18-13-44-53:
dasmith_e6410_server1_8623_v1_33892 dasmith_e6410_server2_8940_v2_45565

./2012-10-18-13-44-53/dasmith_e6410_server1_8623_v1_33892:
config diskstores README.txt restore.sh user

./2012-10-18-13-44-53/dasmith_e6410_server1_8623_v1_33892/config:
cache.xml

./2012-10-18-13-44-53/dasmith_e6410_server1_8623_v1_33892/diskstores:
DEFAULT

./2012-10-18-13-44-53/dasmith_e6410_server1_8623_v1_33892/diskstores/DEFAULT:
dir0

./2012-10-18-13-44-53/dasmith_e6410_server1_8623_v1_33892/diskstores/DEFAULT/dir0:
BACKUPDEFAULT_1.crf BACKUPDEFAULT_1.drf BACKUPDEFAULT.if

./2012-10-18-13-44-53/dasmith_e6410_server1_8623_v1_33892/user:
```

## 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.
- 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:
  1. Duplicate the directory structure of a backed up member for this member.
  2. Rename directories as needed to reflect this member's particular backup, including disk store names.
  3. Clear out all files but the restore script.
  4. Copy in this member's files.
  5. Modify the restore script to work for this member.

## Restore 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.

The restore copies these files back to their original location:

- Disk store files for all stores containing persistent region data.
- 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.

To back up your offline system:

1. Validate, and consider compacting your disk stores before backing them up.
2. Copy all disk store files, and any other files you want to save, to your backup locations.

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.

# Chapter 42

---

## Cache and Region Snapshots

Snapshots allow you to save region data and reload it later. A typical use case is loading data from one environment to another, such as capturing data from a production system and move it into a smaller QA or development system.

In effect, you can load data from one distributed system into another distributed system. Administrators export a snapshot of a region or an entire cache (multiple regions) and later import the snapshot into another region or distributed system by using the `RegionSnapshotService` or `CacheSnapshotService` interface and the `Region.getSnapshotService` or `Cache.getSnapshotService` method.

The snapshot file is a binary file that contains all data from a particular region. The binary format contains serialized key/value pairs and supports PDX type registry to allow the deserialization of PDX data. The snapshot can be directly imported into a region or read entry-by-entry for further processing or transformation into other formats.



**Note:** The previous `Region.loadSnapshot` and `Region.saveSnapshot` APIs have been deprecated. Data written in this format is not compatible with the new APIs.

### Usage and Performance Notes

Optimize the cache and region snapshot feature by understanding how it performs.

#### Cache Consistency and Concurrent Operations

Importing and exporting region data is an administrative operation, and certain simultaneous runtime conditions can cause the import or export operation to fail such as when you are rebalancing partitioned region buckets or experience a network partition event. This behavior is expected, and you should retry the operation. Redoing an export overwrites an incomplete snapshot file, and redoing an import updates partially imported data.

The snapshot feature does not guarantee consistency. Concurrent cache operations during a snapshot import or export can cause data consistency issues. If snapshot consistency is important, VMware recommends that you take your application offline before export and import, to provide a quiet period ensures data consistency in your snapshot.

For example, modifications to region entries during an export can result in a snapshot that contains some but not all updates. If entries { A, B } are updated to { A', B' } during the export, the snapshot can contain { A, B' } depending on the write order. Also, modifications to region entries during an import can cause lost updates in the cache. If the region contains entries { A, B } and the snapshot contains { A', B' }, concurrent updates { A\*, B\* } can result in the region containing { A\*, B' } after the import completes.

The default behavior is to perform all I/O operations on the node where the snapshot operations are invoked. This will involve either collecting or dispersing data over the network if the region is a partitioned region.

## Performance Considerations

When using the data snapshot feature, be aware of the following performance considerations:

- Importing and exporting cache or region snapshots causes additional CPU and network load. You may need to increase CPU capacity or network bandwidth depending on your applications and infrastructure. In addition, if you export regions that have been configured to overflow to disk, you may require additional disk I/O to perform the export.
- When exporting partitioned region data, allocate additional heap memory so the member performing the export can buffer data gathered from other cache members. Allocate at least 10MB per member to your heap in addition to whatever configuration is necessary to support your application or cache.

## Exporting Cache and Region Snapshots

To save GemFire cache or region data to a snapshot that you can later load into another distributed system or region, use the `cache.getSnapshotService.save` API, `region.getSnapshotService.save` API, or the gfsh command-line interface (`export data`).

If an error occurs during export, the export halts and the snapshot operation is canceled. Typical errors that halt an export include scenarios such as full disk, problems with file permissions, and network partitioning.

### Exporting Cache Snapshots

When you export an entire cache, it exports all regions in the cache as individual snapshot files into a directory. If no directory is specified, the default is the current directory. A snapshot file is created for each region, and the export operation automatically names each snapshot filename using the following convention:

```
snapshot-<region>[ -<subregion> ] *
```

When the export operation writes the snapshot filename, it replaces each forward slash ('/') in the region path with a dash ('-').

#### Using Java API:

```
File mySnapshotDir = ...
Cache cache = ...

cache.getSnapshotService().save(mySnapshotDir, SnapshotFormat.GEMFIRE);
```

Optionally, you can set a filter on the snapshot entries during the export. See [Filtering Entries During Import or Export](#) on page 486 for an example.

### Exporting a Region Snapshot

You can also export a specific region using the following API or gfsh command:

#### Java API:

```
File mySnapshot = ...
Region<String, MyObject> region = ...

region.getSnapshotService().save(mySnapshot, SnapshotFormat.GEMFIRE);
```

#### gfsh:

Open a gfsh prompt. After connecting to a GemFire distributed system, at the prompt type:

```
gfsh>export data --region=Region --file=filename.gfd
--member=membername
```

where *Region* corresponds to the name of the region that you want to export, *filename* (must end in .gfd) corresponds to the name of the export file and *membername* corresponds to a member where the region to export is hosted. For example:

```
gfsh>export data --region=region1 --file=region1_2012_10_10.gfd
--member=server1
```

The snapshot file will be written on the remote member at the location specified by the --file argument. For example, in the example command above, the `region1_2012_10_10.gfd` file will be written in the working directory of `server1`. For more information on this command, see [export data](#) on page 691.

## Importing Cache and Region Snapshots

To import a GemFire cache or region data snapshot that you previously exported into another distributed system or region, use the `cache.getSnapshotService.load` API, `region.getSnapshotService.load` API, or the `gfsh` command-line interface (`import data`).

### Import Requirements

Before you import a region snapshot:

- Make sure the cache is configured correctly. Configure all registered PdxSerializers, DataSerializers, and Instantiators; create regions; and ensure the classpath contains any required classes.
- When you import a snapshot containing PDX types, you must wait until the exported type definitions are imported into the cache before inserting data that causes type conflicts. It is recommended that you wait for the import to complete before inserting data.

### Import Limitations

During an import, the `CacheWriter` and `CacheListener` callbacks are not invoked. In addition, the imported data is not distributed to clients or to gateway senders for WAN distribution.

If an error occurs during import, the import is halted and the region will contain some but not all snapshot data.

The state of a cache client is indeterminate after an import. It is likely that the data in the client's cache is inconsistent with the imported data. Take the client offline during the import and restart it after the import completes.

### Importing Cache Snapshots

When you import a cache snapshot, the snapshot file is imported into the same region (match determined by name) that was used during snapshot export. When you import a cache, you import all snapshot files located within a directory into the cache. The API attempts to load all files in the specified directory.

#### Java API:

```
File mySnapshotDir = ...
Cache cache = ...

cache.getSnapshotService().load(mySnapshotDir, SnapshotFormat.GEMFIRE);
```

## Importing a Region Snapshot

### Java API:

```
File mySnapshot = ...
Region<String, MyObject> region = ...

region.getSnapshotService().load(mySnapshot, SnapshotFormat.GEMFIRE);
```

### gfsh:

Open a gfsh prompt. After connecting to a GemFire distributed system, at the prompt type:

```
gfsh>import data --region=Region --file=filename.gfd
--member=membername
```

where *Region* corresponds to the name of the region that you want to import data into; *filename* (must end in .gfd) corresponds to the name of the file to be imported; and *membername* corresponds to the member where the region to be imported is hosted. For example:

```
gfsh>import data --region=region1 --file=region1_2012_10_10.gfd
--member=server2
```

The snapshot file must already reside on the specified member at the location specified in the --file argument before import.

For more information on this command, see [import data](#) on page 696.

## Filtering Entries During Import or Export

You can customize your snapshot by filtering entries during the import or export of a region or a cache.

For example, use filters to limit the export of data to a certain date range. If you set up a filter on the import or export of a cache, the filter is applied to every single region in the cache.

The following example filters snapshot data by even numbered keys.

```
File mySnapshot = ...
Region<Integer, MyObject> region = ...

SnapshotFilter<Integer, MyObject> even = new SnapshotFilter<Integer,
MyObject>() {
    @Override
    public boolean accept(Entry<Integer, MyObject> entry) {
        return entry.getKey() % 2 == 0;
    }
};

RegionSnapshotService<Integer, MyObject> snapsrv =
region.getSnapshotService();
SnapshotOptions<Integer, MyObject> options =
rss.createOptions().setFilter(even);

// only save cache entries with an even key
snapsrv.save(mySnapshot, SnapshotFormat.GEMFIRE, options);
```

## Reading Snapshots Programmatically

You can read a snapshot entry-by-entry for further processing or transformation into other formats.

The following is an example of a snapshot reader that processes entries from a previously generated snapshot file.

```
File mySnapshot = ...  
SnapshotIterator<String, MyObject> iter = SnapshotReader.read(mySnapshot);  
try {  
    while (iter.hasNext()) {  
        Entry<String, MyObject> entry = iter.next();  
  
        String key = entry.getKey();  
        MyObject value = entry.getValue();  
  
        System.out.println(key + " = " + value);  
    }  
} finally {  
    iter.close();  
}
```



# Chapter 43

---

## Network Partitioning

vFabric GemFire architecture and management features help detect and resolve network partition problems.

### How Network Partitioning Management Works

GemFire handles network outages by using a weighting system to determine whether the remaining available members have a sufficient quorum to continue as a distributed system.

Individual members are each assigned a weight, and the quorum is determined by comparing the total weight of currently responsive members to the previous total weight of responsive members.

Your distributed system can split into separate running systems when members lose the ability to see each other. The typical cause of this problem is a failure in the network. To guard against this, vFabric GemFire offers network partitioning detection so that when a failure occurs, only one side of the system keeps running and the other side automatically shuts down.



**Note:** The network partitioning detection feature is only enabled when enable-network-partition-detection is set to true in gemfire.properties. By default, this property is set to false. See [Configure vFabric GemFire to Handle Network Partitioning](#) on page 494 for details. Quorum weight calculations are always performed and logged by the locator regardless of this configuration setting.

The overall process for detecting a network partition is as follows:

1. The distributed system starts up. When you start up a distributed system, you typically start the locators first, then the cache servers and then other members such as applications or processes that access distributed system data.
2. After the locators start up, the oldest locator assumes the role of the membership coordinator. Peer discovery occurs as members come up and locators generate a membership discovery list for the distributed system. Locators hand out the membership discovery list as each member process starts up. This list typically contains a hint on who the current membership coordinator is.
3. Members join and if necessary, depart the distributed system:
  - Member processes make a request to the coordinator to join the distributed system. If authenticated, the coordinator hands the new member the current membership view and begins the process of updating the current view (to add the new member or members) by sending out a view preparation message to existing members in the view.
  - While members are joining the system, it is possible that members are also being removed through the normal failure detection process. Failure detection removes unresponsive or slow members. See [Managing Slow Receivers](#) on page 527 and [Failure Detection and Membership Views](#) on page 490 for descriptions of the failure detection process. If a new membership view is sent out that includes one or more failed processes, the locator will log the new weight calculations. At any point, if quorum loss is detected due to unresponsive processes, the locator will also log a severe level message to identify the failed processes:

Possible loss of quorum detected due to loss of {0} cache processes: {1}  
where {0} is the number of processes that failed and {1} lists the processes.

4. Whenever the coordinator is alerted of a membership change (a member either joins or leaves the distributed system), it generates a new membership view. The membership view is generated by a two-phase protocol:
  - a. In the first phase, the membership coordinator sends out view preparation message to all members and waits 12 seconds for a view preparation ack return message from each member. If the coordinator does not receive an ack message from a member within 12 seconds, the coordinator attempts to connect to the member's failure-detection socket and then attempts to connect to its direct-channel socket. If the coordinator cannot connect to the member's sockets, it declares the member dead and starts the process over again with a new view.
  - b. In the second phase, the coordinator sends out the new membership view to all members that acknowledged the view preparation message. The coordinator waits another 12 seconds for an acknowledgment of receiving the new view from each member. Any members that fail to acknowledge the view are removed from the view.
5. Each time the membership coordinator sends a view, it calculates the total weight of members in the current membership view and compares it to the total weight of the previous membership view. Some conditions to note:
  - When the first membership view is sent out, there are no accumulated losses. The first view only has additions and usually contains the initial locator/coordinator.
  - A new coordinator may have a stale view of membership if it did not see the last membership view sent by the previous (failed) coordinator. If new members were added during that failure, then the new members may be ignored when the first new view is sent out.
  - If members were removed during the failover to the new coordinator, then the new coordinator will have to determine these losses during the view preparation step.
6. If GemFire detects that the total membership weight has dropped by *a configured percentage* within a single membership view change (loss of quorum), then GemFire will declare a network partition event if network partitioning detection has been enabled (`enable-network-partition-detection` set to true). The coordinator sends a network-partitioned-detected UDP message to all members (even to the non-responsive ones) and then closes the distributed system with a `ForcedDisconnectException`. If a member fails to receive the message before the coordinator closes the system, the member is responsible for detecting the event on its own.

The presumption is that when a network partition is declared, the members that can generate a quorum of membership will continue operations and assume the role of the "surviving" side. The surviving members will elect a new coordinator, designate a lead member, and so on.

Note that it is possible for a member to fail during view transmission and that some other process will reuse its fd-sock or direct-channel port, causing a false positive in the member verification step. This is acceptable because it means that the machine that hosted the process is still reachable. No network failure has occurred and the member that did not acknowledge the view preparation message will be removed in a subsequent view.

## Failure Detection and Membership Views

GemFire uses failure detection to remove unresponsive members from membership views.

### Failure Detection

Network partitioning has a failure detection protocol that is not subject to hanging when NICs or machines fail. Failure detection works by detecting missing datagram heartbeats from the peer to the left in the membership view (see "Membership Views" below for the view layout), 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 `gemfire.properties` `ack-wait-threshold` elapses before receiving a response to a message, if a TCP/IP connection cannot be made to the member for peer-to-peer (P2P) messaging, and if no other traffic is detected from the member. For this kind of failure detection, the operator must also have set the `ack-severe-alert-threshold` in `gemfire.properties`.

If a new membership view is sent out that includes one or more failed processes, the locator will log new quorum weight calculations. At any point, if quorum loss is detected due to unresponsive processes, the locator will also log a severe level message to identify the failed processes:

Possible loss of quorum detected due to loss of {0} cache processes: {1}  
where {0} is the number of processes that failed and {1} lists the processes.

## Membership Views

The following is a sample membership view:

```
[info 2012/01/06 11:44:08.164 PST bridgegemfire1 <UDP Incoming Message
Handler> tid=0x1f]
Membership: received new view [ent(5767)<v0>:8700|16]
[ent(5767)<v0>:8700/44876,
ent(5829)<v1>:48034/55334, ent(5875)<v2>:4738/54595,
ent(5822)<v5>:49380/39564,
ent(8788)<v7>:24136/53525]
```

The components of the membership view are as follows:

- The first part of the view (`[ent(5767)<v0>:8700|16]` in the example above) corresponds to the view ID. It identifies:
  - the address and processId of the membership coordinator-- `ent(5767)` in example above.
  - the view-number (`<vXX>`) of the membership view that the member first appeared in-- `<v0>` in example above.
  - membership-port of the membership coordinator-- `8700` in the example above.
  - view-number-- `16` in the example above
- The second part of the view lists all of the member processes in the current view.  
`[ent(5767)<v0>:8700/44876, ent(5829)<v1>:48034/55334,`  
`ent(5875)<v2>:4738/54595, ent(5822)<v5>:49380/39564,`  
`ent(8788)<v7>:24136/53525]` in the example above.
- The overall format of each listed member  
`is:Address(processId)<vXX>:membership-port/distribution port`. The membership coordinator is almost always the first member in the view and the rest are ordered by age.
- The membership-port is the JGroups TCP UDP port that it uses to send datagrams. The distribution-port is the TCP/IP port that is used for cache messaging.
- Each member watches the member to its left for failure detection purposes.

## Membership Coordinators, Lead Members and Member Weighting

Network partition detection uses a designated membership coordinator and a weighting system that accounts for a lead member to determine whether a network partition has occurred.

### Membership Coordinators and Lead Members

The membership coordinator is a member that manages entry and exit of other members of the distributed system. With network partition detection, the coordinator can be any GemFire member but locators are preferred. If all

locators are lost, the system continues to function, but new members will not be able to join until a locator is restarted. After a locator has restarted, the restarted locator will take over the role of coordinator.

When a coordinator is shutting down, it sends out a view that removes itself from list and the other members must determine who the new coordinator is.

The lead member is determined by the coordinator. Any member that has enabled network partition detection, is not hosting a locator, and is not an administrator interface-only member is eligible to be designated as the lead member by the coordinator. The coordinator chooses the longest-lived member that fits the criteria.

The purpose of the lead member role is to provide extra weight. It does not perform any specific functionality.

## Member Weighting System

By default, individual members are assigned the following weights:

- Each member has a weight of 10 except the lead member.
- The lead member is assigned a weight of 15.
- Locators have a weight of 3.

You can modify the default weights for specific members by defining the `gemfire.member-weight` system property upon startup.

The weights of members prior to the view change are added together and compared to the weight of lost members. Lost members are considered members that were removed between the last view and the completed send of the view preparation message. If membership is reduced by a certain percentage within a single membership view change, a network partition is declared.

The loss percentage threshold is 51 (meaning 51%). Note that the percentage calculation uses standard rounding. If loss percentage is equal to or greater than 51%, the membership coordinator initiates shut down.

## Sample Member Weight Calculations

This section provides some example calculations.

**Example 1:** Distributed system with 12 members. 2 locators, 10 cache servers (one cache server is designated as lead member.) View total weight equals 111.

- 4 cache servers become unreachable. Total membership weight loss is 40 (36%). Since 36% is under the 51% threshold for loss, the distributed system stays up.
- 1 locator and 4 cache servers (including the lead member) become unreachable. Membership weight loss equals 48 (43%). Since 43% is under the 51% threshold for loss, the distributed system stays up.
- 5 cache servers (not including the lead member) and both locators become unreachable. Membership weight loss equals 56 (49%). Since 49% is under the 51% threshold for loss, the distributed system stays up.
- 5 cache servers (including the lead member) and 1 locator become unreachable. Membership weight loss equals 58 (52%). Since 52% is greater than the 51% threshold, the coordinator initiates shutdown.
- 6 cache servers (not including the lead member) and both locators become unreachable. Membership weight loss equals 66 (59%). Since 59% is greater than the 51% threshold, the newly elected coordinator (a cache server since 1 locator remains) will initiate shutdown.

**Example 2:** Distributed system with 4 members. 2 cache servers (1 cache server is designated lead member), 2 locators. View total weight is 31.

- Cache server designated as lead member becomes unreachable. Membership weight loss equals 15 or 48%. Distributed system stays up.
- Cache server designated as lead member and 1 locator become unreachable. Member weight loss equals 18 or 58%. Membership coordinator initiates shutdown. If the locator that became unreachable was the membership coordinator, the other locator is elected coordinator and the initiates shutdown.

Even if network partitioning is not enabled, if quorum loss is detected due to unresponsive processes, the locator will also log a severe level message to identify the failed processes:

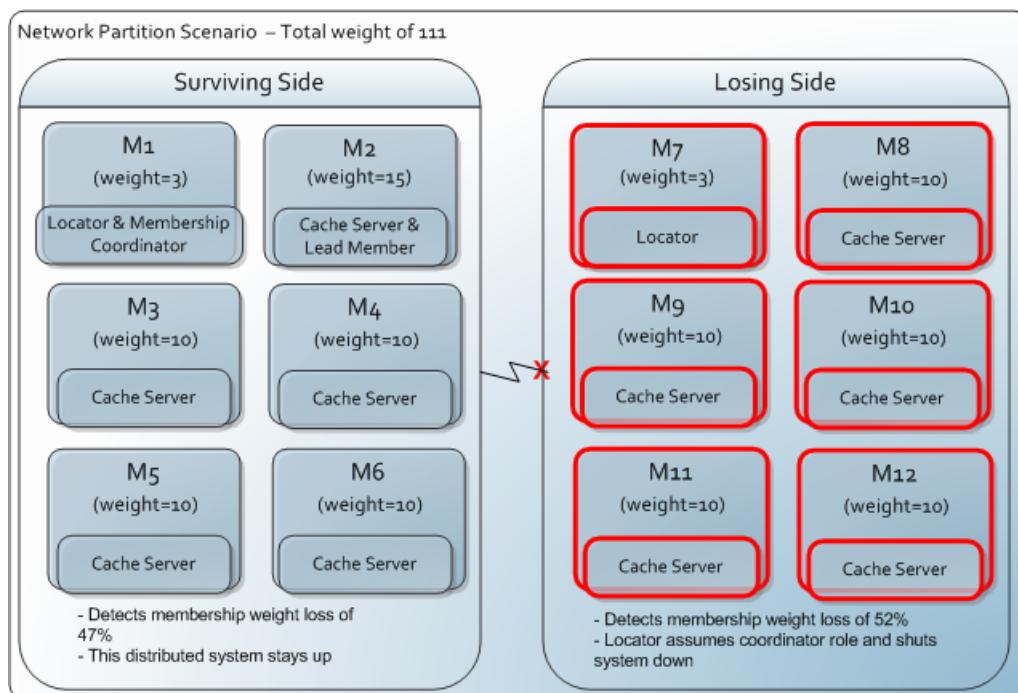
Possible loss of quorum detected due to loss of {0} cache processes: {1}  
where {0} is the number of processes that failed and {1} lists the processes.

Enabling network partition detection allows only one subgroup to survive a split. The rest of the system is disconnected and the caches are closed.

When a shutdown occurs, the members that are shut down will log the following alert message:  
Exiting due to possible network partition event due to loss of {0} cache processes: {1}  
where {0} is the count of lost members and {1} is the list of lost member IDs.

## Network Partitioning Scenarios

This topic describes network partitioning scenarios and what happens to the partitioned sides of the distributed system.



### What the Losing Side Does

In a network partitioning scenario, the "losing side" constitutes the cluster partition where the membership coordinator has detected that there is an insufficient quorum of members to continue.

The membership coordinator calculates membership weight change after sending out its view preparation message. If a quorum of members does not remain after the view preparation phase, the coordinator on the "losing side" declares a network partition event and sends a network-partition-detected UDP message to the members. The coordinator then closes its distributed system with a `ForcedDisconnectException`. If a member fails to receive the message before the coordinator closes the connection, it is responsible for detecting the event on its own.

When the losing side discovers that a network partition event has occurred, all peer members receive a `RegionDestroyedException` with Operation: `FORCED_DISCONNECT`.

If a CacheListener is installed, the afterRegionDestroy callback is invoked with a RegionDestroyedEvent, as shown in this example logged by the losing side. The peer member process IDs are 14291 (lead member) and 14296, and the locator is 14289.

```
[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
```

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

### What Isolated Members Do

When a member is isolated from all locators, it is unable to receive membership view changes. It can't know if the current coordinator is present or, if it has left, whether there are other members available to take over that role. In this condition, a member will eventually detect the loss of all other members and will use the loss threshold to determine whether it should shut itself down. In the case of a distributed system with 2 locators and 2 cache servers, the loss of communication with the non-lead cache server plus both locators would result in this situation and the remaining cache server would eventually shut itself down.

## Configure vFabric GemFire to Handle Network Partitioning

This section lists the configuration steps for network partition detection.

The system uses a combination of locators and system members, designated as lead members, to detect and resolve network partitioning problems.

1. Use locators for member discovery. See [Configuring Peer-to-Peer Discovery](#) on page 173. In addition, use multiple locators.
2. Enable partition detection in the locators and in all system members by setting this in their gemfire.properties:

```
enable-network-partition-detection=true
```

All system members should have the same setting for enable-network-partition-detection. If they don't, the system throws a GemFireConfigException upon startup.

3. Configure regions you want to protect from network partitioning with DISTRIBUTED\_ACK or GLOBAL scope. Do not use DISTRIBUTED\_NO\_ACK scope. The region configurations provided in the region shortcut settings use DISTRIBUTED\_ACK scope. This setting prevents operations from performed throughout the distributed system before a network partition is detected.



**Note:** GemFire issues an alert if it detects distributed-no-ack regions when network partition detection is enabled:

```
Region {0} is being created with scope {1} but
enable-network-partition-detection is enabled in the distributed
system.
```

This can lead to cache inconsistencies if there is a network failure.

4. You must set `enable-network-partition-detection` to true if you are using persistent partitioned regions.
5. These other configuration parameters affect or interact with network partitioning detection. Check whether they are appropriate for your installation and modify as needed.
  - If you have network partition detection enabled, the threshold percentage value for allowed membership weight loss is automatically configured to 51. You cannot modify this value.
  - Failure detection is initiated if a member's `gemfire.properties` `ack-wait-threshold` (default is 15 seconds) and `ack-severe-alert-threshold` (15 seconds) elapses before receiving a response to a message. If you modify the `ack-wait-threshold` configuration value, you should modify `ack-severe-alert-threshold` to match the other configuration value.
  - If the system has clients connecting to it, the clients' `cache.xml` `<cache>` `<pool>` `read-timeout` should be set to at least three times the `member-timeout` setting in the server's `gemfire.properties`. The default `<cache>` `<pool>` `read-timeout` setting is 10000 milliseconds.
  - You can adjust the default weights of members by specifying the system property `gemfire.member-weight` upon startup. For example, if you have some VMs that host a needed service, you could assign them a higher weight upon startup.

## Preventing Network Partitions

This section provides a short list of things you can do to prevent network partition from occurring.

To avoid a network partition:

- Use NIC teaming for redundant connectivity. See [http://www.cisco.com/en/US/docs/solutions/Enterprise/Data\\_Center/vmware/VMware.html#wp696452](http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/vmware/VMware.html#wp696452) for more information.
- It is best if all servers share a common network switch. Having multiple network switches increases the possibility of a network partition occurring. If multiple switches must be used, redundant routing paths should be available, if possible. The weight of members sharing a switch in a multi-switch configuration will determine which partition survives if there is an inter-switch failure.
- In terms of GemFire configuration, consider the weighting of members. For example, you could assign important processes a higher weight.



# Chapter 44

---

## Security

The security framework establishes trust between members and 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.
- A Diffie-Hellman key exchange encrypts sensitive credentials.

### vFabric GemFire Security Features

vFabric GemFire includes a range of built-in authentication and authorization features. It also accommodates security infrastructure plug-ins.

GemFire 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, that 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.

### Security Information Specific to vFabric GemFire

VMware is committed to providing products and solutions that allow you to assess the security of your information, secure your information infrastructure, protect your sensitive information, and manage security information and events to assure effectiveness and regulatory compliance.

As part of this commitment, the following vFabric GemFire-specific security information is provided to help you secure your environment.

Note that the intent of this section is to gather discrete pieces of information in one convenient location to better help you assess and configure the security of your environment. For additional details about the information, see the appropriate sections in this User Guide.

### **External Interfaces, Ports, and Services**

A number of vFabric GemFire processes use either UDP or TCP/IP ports to communicate with other processes or clients.

For example:

- vFabric GemFire members can use multicast to locate and communicate with peer members. You specify multicast addresses and multicast ports in your `gemfire.properties` file or as parameters on the command-line if starting the members using `gfsh`.
- GemFire clients connect to a locator to discover cache servers.
- GemFire clients and servers discover each other on a predefined port (40404 by default) on the localhost.
- JMX clients (such as `gfsh` and JConsole) can connect to JMX Managers and other manageable GemFire members on the pre-defined RMI port 1099. You can configure a different port if necessary.
- Each gateway receiver usually has a port range where it listens for incoming communication.

See [Firewalls and Ports](#) on page 751 for the complete list of ports used by GemFire, their default values, and how to configure them if you do not want to use the default value.

vFabric GemFire does not have any external interfaces or services that need to be enabled or opened.

### **Resources That Must Be Protected**

Certain GemFire configuration files should be readable and writeable *only* by the dedicated user who runs GemFire servers.

- `gemfire.properties`
- `cache.xml`
- `gfsecurity.properties`

The default location of these configuration files is the `defaultConfigs` child directory of the main GemFire installation directory.

### **Log File Locations**

By default, the log files are located in the working directory used when you started the corresponding processes.

For GemFire members (locators and cache servers), you can also specify a custom working directory location when you start each process. See [Logging](#) on page 543 for more details.

The GemFire log files are as follows:

- `locator-name.log`: Contains logging information for the locator process.
- `server-name.log`: Contains logging information for a cache server process.
- `gfsh-YYYY-MM-DD_HH-MM-SS.log`

These log files should be readable and writable *only* by the dedicated user who runs the GemFire servers.

If you contact VMware technical support for help with an issue and the support engineer requests that you submit log files to help resolve the problem, VMware recommends that you remove all sensitive information from the logs before submitting.

## User Accounts Created at Installation

When you install vFabric GemFire on Red Hat Enterprise Linux (RHEL) using the RPM, a user account is automatically created.

This user has the following characteristics:

- ID: `gemfire`
- Group: `vfabric`
- Non-interactive, which means that you cannot directly log in to the RHEL computer as this user. Rather, you must log in as `root` or user with appropriate `sudo` privileges and `su - gemfire`.

The GemFire installation directory is owned by the `gemfire` user, with group `vfabric`.

When you install GemFire on Windows or from a `*.zip` or `*.jar` file, a user account is *not* automatically created for you.

## Obtaining and Installing Security Updates

New versions of vFabric GemFire often include important security fixes, so VMware recommends you keep up to date with the latest releases.

For details about any security fixes in a particular release, see the [release notes](#).

To download the latest `*.zip` or `*.jar` distributions of the vFabric GemFire, go to the [VMware Download Center](#).

For detailed upgrade information, see [Before You Upgrade](#) on page 35.

## Security Implementation Overview

vFabric GemFire can authenticate peer system members, clients, and remote sites. It can also authorize cache operations on a server from clients.

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

1. Use locators for peer discovery within the distributed systems and for client discovery of servers. See [Configuring Peer-to-Peer Discovery](#) on page 173 and [Configuring a Client/Server System](#) on page 183.
2. 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.
3. Implement membership authentication. Depending on your installation and security requirements, you may use a combination of peer-to-peer, client/server, and multi-site settings.
4. 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.
5. If you want to use secure socket layer (SSL) protocol for your peer-to-peer and client/server connections, implement that.

## Where to Place Security Configuration Settings

Any security-related (properties that begin with `security-*`) configuration properties that are normally configured in `gemfire.properties` can be moved to a separate `gfsecurity.properties` file. Placing these configuration settings in a separate file allows you to restrict access to security configuration data. This way, you can still allow read or write access for your `gemfire.properties` file.

Upon startup, GemFire processes will look for the `gfsecurity.properties` file in the following locations in order:

- current working directory
- user's home directory

- classpath

If any password-related security properties are listed in the file but have a blank value, the process will prompt the user to enter a password upon startup.

<b>Related Topics</b>
com.gemstone.gemfire.security

## Where to Place Security Settings

Restrict access to security configuration data by placing related properties (beginning with `security-*`) in a separate file.

You can move security settings that are normally configured in `gemfire.properties` to a separate `gfsecurity.properties` file. This way, you can still allow read or write access for your `gemfire.properties` file.

Upon startup, GemFire processes will look for the `gfsecurity.properties` file in the following locations in order:

- current working directory
- user's home directory
- classpath

If any password-related security properties are listed in the file but have a blank value, the process will prompt the user to enter a password upon startup.

<b>Related Topics</b>
com.gemstone.gemfire.security

## Authentication

A distributed system using authentication bars malicious peers or clients, and deters inadvertent access to its cache.

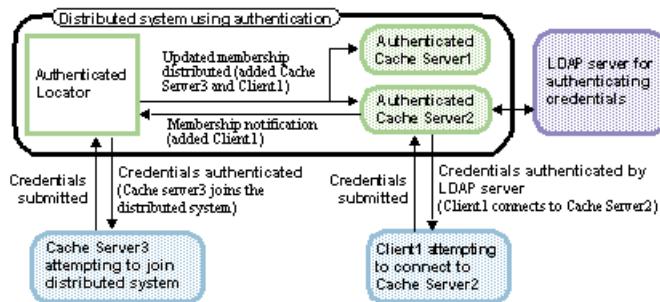
### How Authentication Works

Joining members provide credentials to existing members who check the credentials and either reject the joining member or approve it.

If approved, the connection request returns a `java.security.Principal` object, used to identify the member in future operations.

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

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 an member list that includes itself.



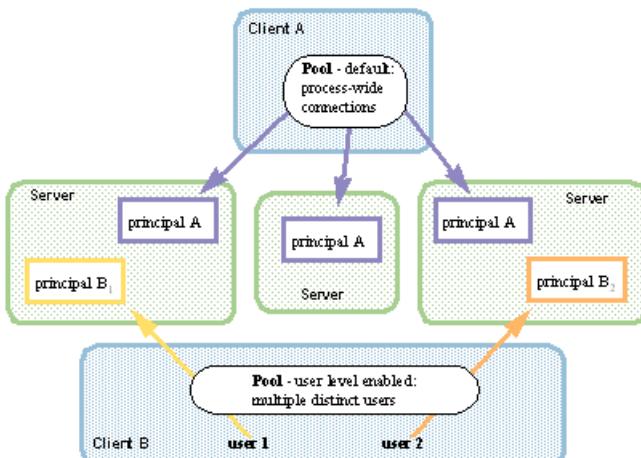
GemFire authentication provides a flexible plug-in framework. Any security infrastructure can be plugged in to the system as long as the plug-ins implement the required GemFire interfaces.

## Client Authentication Options

The GemFire client can connect in two different ways:

1. **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.
2. **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.



## 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.

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

## When a Member Fails to Join

The following describe the scenarios that occur 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 `DistributedSystem.connect` method 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 client handshake 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.

## Implementing Authentication

vFabric 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.

## Procedure

Use the following procedure to implement authentication in your GemFire application:

1. Determine the method of authentication, such as LDAP or PKCS, that you will use. It is assumed that you know how to use it.

For some examples of implementing authentication, see the following code samples, included with your GemFire installation:

- <install directory>/SampleCode/quickstart/quickstart/SecurityClient.java
- <install directory>/SampleCode/quickstart/quickstart/SecurityServer.java
- <install directory>/SampleCode/quickstart/quickstart/MultiuserSecurityClient.java
- <install directory>/SampleCode/quickstart/quickstart/MultiuserSecurityServer.java

The <install directory>/templates subdirectory also contains Java code you may find useful when implementing authentication.

2. Determine any special properties required for your authentication's credentials initialization and 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 these:
  - Pass the additional properties through the `gemfire.properties` file (or `gfsecurity.properties` file if you are creating a special restricted access file for security configuration) settings or programmatically, using the `set` methods in the `ClientCacheFactory`, before the call to the `create` method. All properties starting with `security-` are automatically passed to the `AuthInitialize` implementation.
  - Obtain the properties dynamically in the `AuthInitialize.getCredentials` method.
3. For joining members, program and configure the credentials initialization plug-in:
  - a. Create an implementation of the GemFire `com.gemstone.gemfire.security.AuthInitialize` interface.
    - a. Program a public static method to return an instance of the class.
    - b. Program the `getCredentials` method to create all properties required by the `Authorize` method via the member's credentials.
  - b. For peers and locators, set the `gemfire.properties` (or `gfsecurity.properties` file if you are creating a special restricted access file for security configuration) `security-peer-auth-init` to the fully qualified name of the static method you programmed that returns an instance of the class. In these examples, the method is named `create`. Example:
 

```
//Peer init example where myAuthInitImpl.create returns the
//instance of AuthInitialize
security-peer-auth-init=myAuthPkg.myAuthInitImpl.create
```
  - c. For clients, set the `gemfire.properties` (or `gfsecurity.properties` file if you are creating a special restricted access file for security configuration) `security-client-auth-init` to the fully qualified name of the method you programmed that returns an instance of the `AuthInitialize` class. Example:
 

```
//Client/WAN init example where myAuthInitImpl.create returns
//the instance of AuthInitialize
security-client-auth-init=myAuthPkg.myAuthInitImpl.create
```
  - d. For all members, set any additional `gemfire.properties` (or `gfsecurity.properties` file if you are creating a special restricted access file for security configuration) `security-*` properties required by your `AuthInitialize` implementation.
4. For authorizing members, program and configure the credentials authorization plug-in:
  - a. Implement the GemFire `com.gemstone.gemfire.security.Authenticator` interface:
    - a. Program a public static, zero-argument method to return an instance of the class.

- b. Program the `authenticate` method to authenticate the credentials and return a `java.security.Principal` object.
  - b. For peers and locators set the `gemfire.properties` (or `gfsecurity.properties`) file if you are creating a special restricted access file for security configuration) `security-peer-authenticator` to the fully qualified name of the method that returns an instance of the Authenticator class. Example:
 

```
//Peer auth example where myAuthenticatorImpl.create
//returns the instance of Authenticator
security-peer-authenticator=myAuthPkg.myAuthenticatorImpl.create
```
  - c. For servers, set the `gemfire.properties` (or `gfsecurity.properties`) file if you are creating a special restricted access file for security configuration) `security-client-authenticator` to the fully qualified name of the method that returns an instance of the Authenticator class. Example:
 

```
//Client/WAN auth example where myAuthenticatorImpl.create
//returns the instance of Authenticator
security-client-authenticator=myAuthPkg.myAuthenticatorImpl.create
```
  - d. For all members, set any additional `gemfire.properties` (or `gfsecurity.properties`) file if you are creating a special restricted access file for security configuration) `security-*` properties required by your Authenticator implementation.
5. For all members, provide the list of authenticated locators in the `gemfire.properties`.

### **Locators That Require Authentication**

Co-located locators, such as those started with the LocatorLauncher API, do not require security settings because they do not join the distributed system as individual members.

All other standalone locators, including those started with the `gfsh start locator` command must be configured with the correct security settings.

### **Setting Up JMX Authentication for GemFire Management and Monitoring**

To force JMX clients such as `gfsh` and GemFire Pulse to authenticate into the GemFire management system, you must configure the JMX Manager node.

By default, the JMX manager allows clients without credentials to connect. To set up JMX authentication for the management system:

1. Verify that the `jmx-manager` GemFire property is set to true on any node that you want to be able to become a JMX Manager and authenticate clients. If this property is set to false or not specified, then all other `jmx-manager-*` properties are ignored.
2. Create a password file that contains entries for the user names and passwords you want to grant access to GemFire's management and monitoring system. For example:

```
#the gemfiremonitor user has password Abc!@#
#the gemfiremanager user has password 123Gh2!

gemfiremonitor Abc!@#
gemfiremanager 123Gh2!
```

3. On each of your JMX Manager-enabled nodes, set the GemFire property `jmx-manager-password-file` to the name of the file you created in step 2. This will require clients to authenticate when connecting to a JMX Manager node in GemFire.
4. If you wish to further restrict access to system operations, you can also set up an access file for the JMX Manager. The access file indicates whether the users listed in the password file have the ability to read system

MBeans (monitor the system) or whether they can additionally modify MBeans (perform operations). For example, you can define the following:

```
#the gemfiremonitor user has readonly access
#the gemfiremanager user has readwrite access

gemfiremonitor readonly
gemfiremanager readwrite
```

5. On each of your JMX Manager-enabled nodes, set the GemFire property `jmx-manager-access-file` to the name of the file you created in step 4. This will associate MBean permissions to the users who authenticate to the JMX Manager node in GemFire.
6. If desired, enable SSL for your JMX Manager connections. To enable SSL, make sure the `jmx-manager-port` property is set to a non-zero value and set the `jmx-manager-ssl` property to true. Then configure all other SSL-related GemFire properties as described in [Implementing SSL](#) on page 513.



**Note:** The GemFire property `ssl-enabled` does not apply to the JMX Manager. Use the `jmx-manager-ssl` property instead.

For more information on the format of the password and access file, see <http://docs.oracle.com/javase/6/docs/technotes/guides/management/agent.html>.

### Encrypting Passwords for Use in cache.xml

vFabric GemFire provides a gfsh utility to generate encrypted passwords.

You may need to specify an encrypted password in `cache.xml` when configuring JNDI connections to external JDBC data sources. See [Configuring Database Connections Using JNDI](#) on page 404 for configuration examples.

The `cache.xml` file accepts passwords in clear text or encrypted text.

To generate an encrypted password, use the [encrypt password](#) on page 689 command in gfsh. The following example shows a sample command invocation and output (assuming `my_password` is the actual password for the data source). After you [Starting gfsh](#) on page 641, enter the following command:

```
gfsh>encrypt password --password=my_password
AB80B8E1EE8BB5701D0366E2BA3C3754
```

Copy the output from the gfsh command to the `cache.xml` file as the value of the `password` attribute of the `jndi-binding` tag embedded in `encrypted()`, just like a method parameter. Enter it as encrypted, in this format:

```
password="encrypted(83f0069202c571faf1ae6c42b4ad46030e4e31c17409e19a)"
```

To use a non-encrypted (clear text) password, put the actual password as the value of the `password` attribute of the `jndi-binding` tag, like this:

```
password="password"
```

### Encrypt 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 - not peer-to-peer 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://download.oracle.com/javase/1.5.0/docs/guide/security/CryptoSpec.html#AppA>.

Before you begin, you need to understand how to use your security algorithm.

### Enable Server Authentication of Client with Diffie-Hellman

Set this in property in the client's `gemfire.properties` (or `gfsecurity.properties` file if you are creating a special restricted access file for security configuration):

- **`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.

### Enable Client Authentication of Server

This requires server authentication of client with Diffie-Hellman to be enabled. To have your client authenticate its servers, in addition to being authenticated:

1. In server `gemfire.properties` (or `gfsecurity.properties` file if you are creating a special restricted access file for security configuration), set:
  - a. **`security-server-kspath`** . Path of the PKCS#12 keystore containing the private key for the server
  - b. **`security-server-ksalias`** . Alias name for the private key in the keystore.
  - c. **`security-server-kspasswd`** . Keystore and private key password, which should match.
2. In client `gemfire.properties` (or `gfsecurity.properties` file if you are creating a special restricted access file for security configuration), set:
  - a. **`security-client-kspasswd`** . Password for the public key file store on the client
  - b. **`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

### Set the Key Size for AES and Blowfish Encryption Keys

For algorithms like AES, especially if large key sizes are used, 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 in combination with challenge-response from server to client to prevent replay and other types of 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, set the key size between 128 and 448 bits, inclusive.

### 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 API documentation.



**Note:** 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. VMware 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 (or `gfsecurity.properties` file if you are creating a special restricted access file for security configuration) settings for 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 the `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 (or `gfsecurity.properties` file if you are creating a special restricted access file for security configuration) settings for client and server.

- Client:

```
security-client-auth-init=templates.security.PKCSAuthInit.create
security-keystorepath="keystore path"
security-alias="alias"
security-keystorerpass="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 the `gemfire.properties` (or `gfsecurity.properties` file if you are creating a special restricted access file for security configuration).

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` also gets the password for the keystore file from the `security-keystorerpass` 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://download.oracle.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 -genkeypair \
-dname "cn=Your Name, ou=GemFire, o=GemStone, c=US" \
-storetype PKCS12 \
-keyalg RSA \
-keysize 2048 \
-alias gemfire8 \
-keystore gemfire8.keystore \
-storepass your_password \
-validity 180
```

This step creates a keystore called `gemfire8.keystore` in the local directory and adds a public/private key pair to it.

2. Export the self-signed certificate:

```
keytool -exportcert \
-storetype PKCS12 \
-keyalg RSA
-keysize 2048
-alias gemfire8 \
-keystore gemfire8.keystore \
-storepass your_password
```

```
-rfc \
-file gemfire8.cer
```

After successfully exporting the certificate, you should see the following message:

```
Certificate stored in file <gemfire8.cer>
```

The above commands exports the certificate file to the current local directory.

3. Import the signed certificate to the truststore:

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

You will be prompted to enter your keystore password. After you have authenticated, the certificate appears. After you respond to the prompt "Trust this certificate?", the certificate is imported into the certificatetruststore keystore, creating it if necessary.

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.

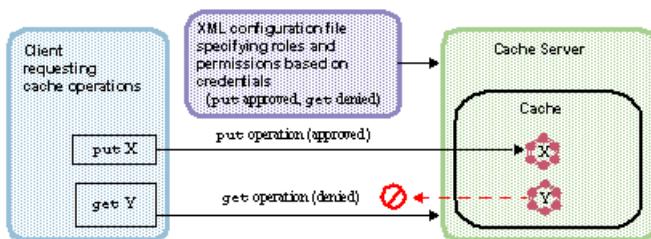
## Authorization

Client operations on a cache server can be restricted or completely blocked based on the roles and permissions assigned to the credentials submitted by the client.

### 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.



The principal, which you associate with the client when it is authenticated, is used by the authorization plug-in to allow or disallow each 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`.



**Note:** 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 all notifications can also be authorized in the post-operation phase where the callback can peek and even modify the result being sent out.

### **Implementing Authorized Access Control for the Cache**

To use authorization for client/server systems, your client connections must be authenticated by their servers.

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:
  - a. Create an implementation of the GemFire `com.gemstone.gemfire.security.AccessControl` interface
    - a. Program a public static method to return an instance of the class.
    - b. 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.



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

- c. Program the `authorizeOperation` method to perform whatever pre- and post-operation authorization activities required by your application. The `OperationContext` has the `OperationCode` and a boolean indicating whether the call is pre-operation or post-operation. 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.
- b. Set the `gemfire.properties` 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. Example:
 

```
//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 myAccessControl.create returns the
// instance of AuthInitialize
security-client-accessor-pp=myAuthPkg.myAccessControl.create
```

Your `authorizeOperation` method will be invoked before and/or after each client operation, as configured.

### **Performance and Programming Considerations**

This section provides guidelines for improving the efficiency of the authorization process.

#### **Performance**

For each new connection request from the client, the system authenticates the connection, and instantiates the callback for authorization of data requests and data updates coming in on that connection. 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

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, 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.

## Authorization Example

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



**Note:** 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. VMware 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 (or `gfsecurity.properties` file if you are creating a special restricted access file for security configuration) 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 security templates 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 topic lists an example XML file for the dtd. The example defines 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`. On the functions, `optimizeForWrite` must be `false` and `keySet` must be `KEY-0` and `KEY-1`.

```
<!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>

```

## SSL

SSL protects your data in transit between applications.

### How SSL Works

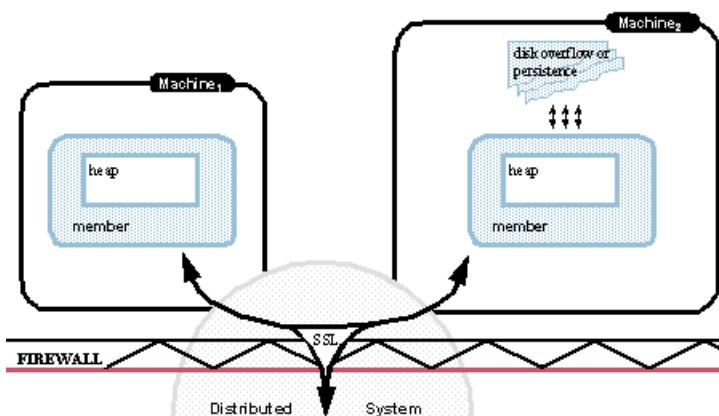
For in transit data, SSL ensures that only the applications identified by you can share distributed system data.

To be secure, the data that is cached in a vFabric GemFire 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 relies on your standard system security features such as firewalls, operating system settings, and JDK security settings.

The SSL implementation ensures that only the applications identified by you can share distributed system data in transit. 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 system members inside and outside the firewalls.



### Implementing SSL

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

You configure SSL for mutual authentication between members and to protect your data during distribution. If configured, SSL is used for all stream-socket communication. GemFire uses SSL connections from the Java Secure Sockets Extension (JSSE) package.

1. Make sure your Java installation includes the JSSE API and familiarize yourself with its use. For information, see the Oracle JSSE website <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html>.
2. Configure your security provider:
  - a. 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.
  - b. Specify provider-required configuration settings. These are usually keystore and truststore configurations. Your provider documentation should include specific configuration requirements. You can add these configurations in a separate, restricted-access `gfsecurity.properties` file.
3. Configure your distributed system members for SSL:
  - a. Use locators for member discovery within the distributed systems and for client discovery of servers. See [Configuring Peer-to-Peer Discovery](#) on page 173 and [Configuring a Client/Server System](#) on page 183.
  - b. Configure all system members for SSL communication. See SSL properties (ssl-\*) in [gemfire.properties and gfsecurity.properties \(GemFire Properties\)](#) on page 851. 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 `ssl-ciphers` to one of these three lines:

```
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
```

## SSL Sample Implementation

A simple example demonstrates 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.

## gemfire.properties File

You can enable SSL in the `gemfire.properties` file:

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

## gfsecurity.properties File

You can specify the provider-specific settings in `gfsecurity.properties` file, which can then be secured by restricting access to this file. The following example configures the default JSSE provider settings included with the JDK.

```
javax.net.ssl.keyStoreType=jks  
javax.net.ssl.keyStore=/path/to/trusted.keystore  
javax.net.ssl.keyStorePassword=password  
javax.net.ssl.trustStore=/path/to/trusted.keystore  
javax.net.ssl.trustStorePassword=  
security-username=xxxx  
security-userPassword=yyyy
```

## Locator Startup

Before starting other system members, we started the locator with the SSL and provider-specific configuration settings. After properly configuring `gemfire.properties` and `gfsecurity.properties`, start the locator and provide the location of the properties files. If any of the password fields are left empty, you will be prompted to enter a password.

```
gfsh>start locator --name=my_locator --port=12345 \  
--properties-file=/path/to/your/gemfire.properties  
--security-properties-file=/path/to/your/gfsecurity.properties
```

## Other Member Startup

Applications and cache servers can be started similarly to the locator startup, with the appropriate `gemfire.properties` file and `gfsecurity.properties` files placed in the current working directory. You can also pass in the location of both files as system properties on the command line. For example:

```
gfsh>start server --name=my_server \  
--properties-file=/path/to/your/gemfire.properties \  
--security-properties-file=/path/to/your/gfsecurity.properties
```



# Chapter 45

---

## Performance Tuning and Configuration

A collection of tools and controls allow you to monitor and adjust vFabric GemFire performance.

### Improving GemFire Performance on vSphere

This topic provides guidelines for tuning vSphere virtualized environments that host vFabric GemFire deployments.

Without tuning, GemFire can suffer a performance drop in virtual environments, including the VMWare vSphere virtual platform. You can expect to see significant performance degradation when running GemFire on vSphere versus running GemFire on dedicated hardware.

vFabric GemFire has been certified and tested on vSphere 5.0.

#### Operating System Guidelines

Use the latest supported version of the guest OS, and use Java large paging.

- **Use the latest supported version of the guest operating system.** This guideline is probably the most important. Upgrade the guest OS to a recent version supported by GemFire. For example, for RHEL, use at least version 6.0 or for SLES, use at least 11.0. For Windows, use Windows Server 2008. For RedHat Linux users, it is particularly beneficial to use RHEL 6 since there are specific enhancements in the RHEL 6 release that improve virtualized latency sensitive workloads. See [vFabric GemFire Supported Configurations](#) on page 3 for a list of supported operating system versions.
- **Use Java large paging in guest OS.** Configure Java on the guest OS to use large pages. Add the following command line option when launching Java:

```
-XX:+UseLargePages
```

#### NUMA, CPU, and BIOS Settings

This section provides VMware- recommended NUMA, CPU, and BIOS settings for your hardware and virtual machines.

- Always enable hyper-threading, and do not overcommit CPU.
- For most production vFabric GemFire servers, always use virtual machines with at least two vCPUs .
- Apply non-uniform memory access (NUMA) locality by sizing virtual machines to fit within the NUMA node.
- VMware recommends the following BIOS settings:
  - **BIOS Power Management Mode:** Maximum Performance.
  - **CPU Power and Performance Management Mode:** Maximum Performance.
  - **Processor Settings:**Turbo Mode enabled.
  - **Processor Settings:**C States disabled.



**Note:** Settings may vary slightly depending on your hardware make and model. Use the settings above or equivalents as needed.

## Physical and Virtual NIC Settings

These guidelines help you reduce latency.

- **Physical NIC:** VMware recommends that you disable interrupt coalescing on the physical NIC of your ESXi host by using the following command:

```
ethtool -C vmnicX rx-usecs 0 rx-frames 1 rx-usecs-irq 0 rx-frames-irq 0
```

where `vmnicX` is the physical NIC as reported by the ESXi command:

```
esxcli network nic list
```

You can verify that your settings have taken effect by issuing the command:

```
ethtool -C vmnicX
```

If you restart the ESXi host, the above configuration must be reapplied.



**Note:** Disabling interrupt coalescing can reduce latency in virtual machines; however, it can impact performance and cause higher CPU utilization. It can also defeat the benefits of Large Receive Offloads (LRO) because some physical NICs (such as Intel 10GbE NICs) automatically disable LRO when interrupt coalescing is disabled. See <http://kb.vmware.com/kb/1027511> for more details.

- **Virtual NIC:** Use the following guidelines when configuring your virtual NICs:

- Use VMXNET3 virtual NICs for your latency-sensitive or otherwise performance-critical virtual machines. See <http://kb.vmware.com/kb/1001805> for details on selecting the appropriate type of virtual NIC for your virtual machine.
- VMXNET3 supports adaptive interrupt coalescing that can help drive high throughput to virtual machines that have multiple vCPUs with parallelized workloads (multiple threads), while minimizing latency of virtual interrupt delivery. However, if your workload is extremely sensitive to latency, VMware recommends that you disable virtual interrupt coalescing for your virtual NICs. You can do this programmatically via API or by editing your virtual machine's .vmx configuration file. Refer to your vSphere API Reference or VMware ESXi documentation for specific instructions.

## VMware vSphere vMotion and DRS Cluster Usage

This topic discusses use limitations of vSphere vMotion, including the use of it with DRS.

- When you first commission the data management system, place VMware vSphere Distributed Resource Scheduler™ (DRS) in manual mode to prevent an automatic VMware vSphere vMotion® operation that can affect response times.
- Reduce or eliminate the use of vMotion to migrate GemFire virtual machines when they are under heavy load.
- Do not allow vMotion migrations with vFabric GemFire locator processes, as the latency introduced to this process can cause other members of the vFabric GemFire servers to falsely suspect that other members are dead.
- Use dedicated vFabric GemFire vSphere DRS clusters. This is especially important when you consider that the physical NIC and virtual NIC are specifically tuned to disable Interrupt Coalescing on every NIC of an ESXi host in the cluster. This type of tuning benefits vFabric GemFire workloads, but it can hurt other non-vFabric GemFire workloads that are memory throughput-bound as opposed to latency sensitive as in the case of vFabric GemFire workloads.
- If using a dedicated vSphere DRS cluster is not an option, and vFabric GemFire must run in a shared DRS cluster, make sure that DRS rules are set up not to perform vMotion migrations on vFabric GemFire virtual machines.

- If you must use vMotion for migration, VMware recommends that all vMotion migration activity of vFabric GemFire members occurs over 10GbE, during periods of low activity and scheduled maintenance windows.

## **Placement and Organization of Virtual Machines**

This section provides guidelines on JVM instances and placement of redundant copies of cached data.

- Have one JVM instance per virtual machine.
- Increasing the heap space to service the demand for more data is better than installing a second instance of a JVM on a single virtual machine. If increasing the JVM heap size is not an option, consider placing the second JVM on a separate newly created virtual machine, thus promoting more effective horizontal scalability. As you increase the number of vFabric GemFire servers, also increase the number of virtual machines to maintain a 1:1:1 ratio among the vFabric GemFire server, the JVM, and the virtual machines.
- Size for a minimum of four vCPU virtual machines with one vFabric GemFire server running in one JVM instance. This allows ample CPU cycles for the garbage collector, and the rest for user transactions.
- Because vFabric GemFire can place redundant copies of cached data on any virtual machine, it is possible to inadvertently place two redundant data copies on the same ESX/ESXi host. This is not optimal if a host fails. To create a more robust configuration, use VM1-to-VM2 anti-affinity rules, to indicate to vSphere that VM1 and VM2 can never be placed on the same host because they hold redundant data copies.

## **Virtual Machine Memory Reservation**

This section provides guidelines for sizing and setting memory.

- Set memory reservation at the virtual machine level so that ESXi provides and locks down the needed physical memory upon virtual machine startup. Once allocated, ESXi does not allow the memory to be taken away.
- Do not overcommit memory for vFabric GemFire hosts.
- When sizing memory for a vFabric GemFire server within one JVM on one virtual machine, the total reserved memory for the virtual machine should not exceed what is available within one NUMA node for optimal performance.

## **vSphere High Availability and vFabric GemFire**

On vFabric GemFire virtual machines, disable vSphere High Availability (HA).

If you are using a dedicated vFabric GemFire DRS cluster, then you can disable HA across the cluster. However, if you are using a shared cluster, exclude vFabric GemFire virtual machines from vSphere HA.

Additionally, to support high availability, you can also set up anti-affinity rules between the vFabric GemFire virtual machines to prevent two vFabric GemFire servers from running on the same ESXi host within the same DRS cluster.

## **Storage Guidelines**

This section provides storage guidelines for persistence files, binaries, logs, and more.

- Use the PVSCSI driver for I/O intensive vFabric GemFire workloads.
- Align disk partitions at the VMFS and guest operating system levels.
- Provision VMDK files as eagerzeroedthick to avoid lazy zeroing for vFabric GemFire members.
- Use separate VMDKs for vFabric GemFire persistence files, binaries, and logs.
- Map a dedicated LUN to each VMDK.
- For Linux virtual machines, use NOOP scheduling as the I/O scheduler instead of Completely Fair Queuing (CFQ). Starting with the Linux kernel 2.6, CFQ is the default I/O scheduler in many Linux distributions. See <http://kb.vmware.com/kb/2011861> for more information.

## **Additional Resources**

VMware provides additional resources for optimizing vSphere, Java applications, and vFabric GemFire.

- "Performance Best Practices for VMware vSphere 5.0" -  
[http://www.vmware.com/pdf/Perf\\_Best\\_Practices\\_vSphere5.0.pdf](http://www.vmware.com/pdf/Perf_Best_Practices_vSphere5.0.pdf)
- "Best Practices for Performance Tuning of Latency-Sensitive Workloads in vSphere Virtual Machines" -  
<http://www.vmware.com/files/pdf/techpaper/VMW-Tuning-Latency-Sensitive-Workloads.pdf>
- "Enterprise Java Applications on VMware - Best Practices Guide" -  
<http://www.vmware.com/resources/techresources/1087>
- "High Performance Data with VMware vFabric™ GemFire® Best Practices Guide" -  
<https://www.vmware.com/files/pdf/techpaper/vmw-vfabric-gemFire-best-practices-guide.pdf>

## Performance Controls

This topic provides tuning suggestions of particular interest to developers, primarily programming techniques and cache configuration.

Before you begin, you should understand vFabric GemFire [Basic Configuration and Programming](#) on page 131.

### Data Serialization

In addition to standard Java serialization, GemFire offers serialization options that give you higher performance and greater flexibility for data storage, transfers, and language types.

Under *Developing with vFabric GemFire*, see [Data Serialization](#) on page 257.

### Setting Cache Timeouts

Cache timeout properties are declared in the `cache.xml` file and can also be set through methods of the interface, `com.gemstone.gemfire.cache.Cache`.

- `search-timeout`. Specifies how long a `netSearch` operation can wait for data before timing out. The default is 5 minutes. You may want to change this based on your knowledge of the network load or other factors.

The next two attributes describe timeout settings for locking in regions with global scope. Locking operations can time out in two places: when waiting to obtain a lock (lock time out); and when holding a lock (lock lease time). Operations that modify objects in a global region use automatic locking. In addition, you can manually lock a global region and its entries through `com.gemstone.gemfire.cache.Region`. The explicit lock methods provided by the APIs allow you to specify a lock timeout parameter. The lock time out for implicit operations and the lock lease time for implicit and explicit operations are governed by these cache-wide parameter settings:

- `lock-timeout`. Timeout for object lock requests. The setting affects automatic locking only, and does not apply to manual locking. The default is 1 minute. If a lock request does not return before the specified timeout period, it is cancelled and returns with a failure.
- `lock-lease`. Timeout for object lock leases. The setting affects both automatic locking and manual locking. The default is 2 minutes. Once a lock is obtained, it may remain in force for the lock lease time period before being automatically cleared by the system.

### Controlling Socket Use

For peer-to-peer communication, you can manage socket use at the system member level and at the thread level.

The `conserve-sockets` setting indicates whether application threads share sockets with other threads or use their own sockets for distributed system member communication. This setting has no effect on communication between a server and its clients, but it does control the server's communication with its peers or a gateway sender's communication with a gateway receiver. In client/server settings in particular, where there can be a large number of clients for each server, controlling peer-to-peer socket use is an important part of tuning server performance.

You configure `conserve-sockets` for the member as a whole in `gemfire.properties`. Additionally, you can change the sockets conservation policy for the individual thread through the API.

When `conserve-sockets` is set to false, each application thread uses a dedicated thread to send to each of its peers and a dedicated thread to receive from each peer. Disabling socket conservation requires more system resources, but can potentially improve performance by removing socket contention between threads and optimizing distributed ACK operations. For distributed regions, the put operation, and destroy and invalidate for regions and entries, can all be optimized with `conserve-sockets` set to false. For partitioned regions, setting `conserve-sockets` to false can improve general throughput.



**Note:** When you have transactions operating on EMPTY, NORMAL or PARTITION regions, make sure that `conserve-sockets` is set to false to avoid distributed deadlocks.

You can override the `conserve-sockets` setting for individual threads. These methods are in `com.gemstone.gemfire.distributed.DistributedSystem`:

- `setThreadsSocketPolicy`. Sets the calling thread's individual socket policy, overriding the policy set for the application as a whole. If set to true, the calling thread shares socket connections with other threads. If false, the calling thread has its own sockets.
- `releaseThreadsSockets`. Frees any sockets held by the calling thread. Threads hold their own sockets only when `conserve-sockets` is false. Threads holding their own sockets can call this method to avoid holding the sockets until the socket-lease-time has expired.

A typical implementation might set `conserve-sockets` to true at the application level and then override the setting for the specific application threads that perform the bulk of the distributed operations. The example below shows an implementation of the two API calls in a thread that performs benchmark tests. The example assumes the class implements `Runnable`. Note that the invocation, `setThreadsSocketPolicy(false)`, is only meaningful if `conserve-sockets` is set to true at the application level.

```
public void run() {
    DistributedSystem.setThreadsSocketPolicy(false);
    try {
        // do your benchmark work
    } finally {
        DistributedSystem.releaseThreadsSockets();
    }
}
```

## Management of Slow Receivers

You have several options for handling slow members that receive data distribution.



**Note:** The slow receiver options control only to peer-to-peer communication between distributed regions using TCP/IP. This topic does not apply to client/server or multi-site communication, or to communication using the UDP unicast or IP multicast protocols.

Most of the options for handling slow members are related to on-site configuration during system integration and tuning. For this information, see [Slow Receivers with TCP/IP](#) on page 525.

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.



**Note:** 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 large enough for the objects you distribute. The socket buffer size is set using `gemfire.socket-buffer-size`.

By default, distribution between system members is performed synchronously. With synchronous communication, when one member is slow to receive, it can cause its producer members to slow down as well. This can lead to general performance problems in the distributed system.

The specifications for handling slow receipt primarily affect how your members manage distribution for regions with distributed-no-ack scope, but it can affect other distributed scopes as well. If no regions have distributed-no-ack scope, this mechanism is unlikely to kick in at all. When slow receipt handling does kick in, however, it affects all distribution between the producer and consumer, regardless of scope. 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.

### Configuration Options

The slow receiver options are set in the producer member's region attribute, enable-async-conflation, and in the consumer member's `async* gemfire.properties` settings.

### Delivery Retries

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 info message when the delivery finally succeeds.

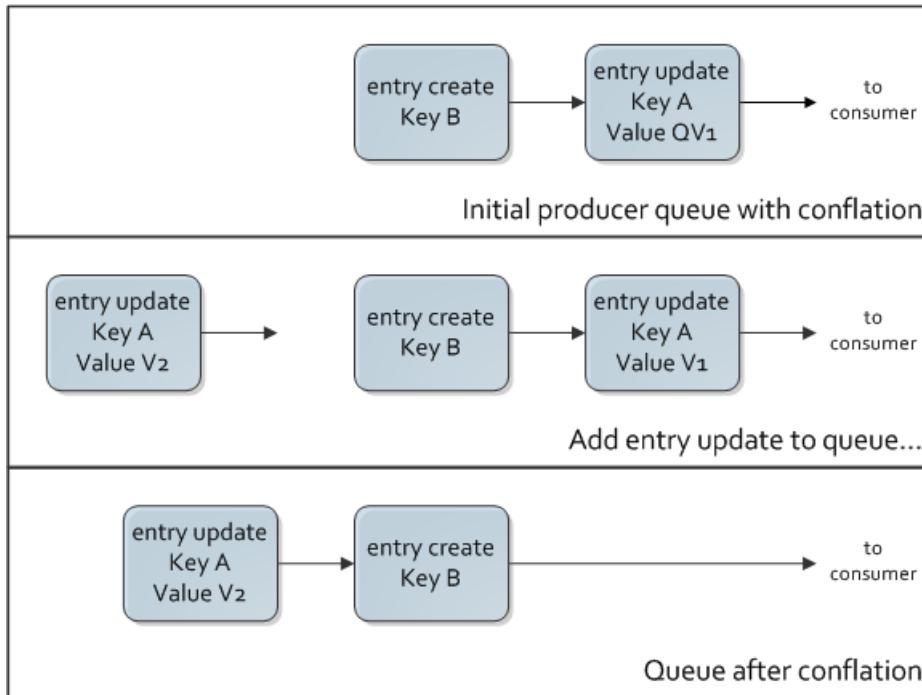
### Asynchronous Queueing For Slow Receivers

Your consumer members can be configured so that their producers switch to asynchronous messaging if the consumers are slow to respond to cache message distribution.

When a producer switches, it creates a queue to hold and manage that consumer's cache messages. When the queue empties, the producer switches back to synchronous messaging for the consumer. The settings that cause the producers to switch are specified on the consumer side in `gemfire.properties` file settings.

If you configure your consumers for slow receipt queuing, and your region scope is distributed-no-ack, you can also configure the producer to conflate entry update messages in its queues. This configuration option is set as the region attribute `enable-async-conflation`. By default distributed-no-ack entry update messages are not conflated.

Depending on the application, conflation can greatly reduce the number of messages the producer needs to send to the consumer. With conflation, when an entry update is added to the queue, if the last operation queued for that key is also an update operation, the previously enqueued update is removed, leaving only the latest update to be sent to the consumer. Only entry update messages originating in a region with distributed-no-ack scope are conflated. Region operations and entry operations other than updates are not conflated.



Some conflation may not occur because entry updates are sent to the consumer before they can be conflated. For this example, assume no messages are sent while the update for Key A is added.



**Note:** This method of conflation is different from that used for conflating gateway sender queues in multi-site installations. However, this method behaves the same as server-to-client conflation.

You can enable queue conflation on a region-by-region basis. You should always enable it unless it is incompatible with your application needs. Conflation reduces the amount of data queued and distributed.

These are reasons why conflation might not work for your application:

- With conflation, earlier entry updates are removed from the queue and replaced by updates sent later in the queue. This is problematic for applications that depend on a specific ordering of entry modifications. For example, if your receiver has a CacheListener that needs to know about every state change, you should disable conflation.
- If your queue remains in use for a significant period and you have entries that are updated frequently, you could have a series of update message replacements resulting in a notable delay in the arrival of any update for some entries. Imagine that update 1, before it is sent, is removed in favor of a later update 2. Then, before update 2 can be sent, it is removed in favor of update 3, and so on. This could result in unacceptably stale data on the receiver.

### Increasing the Ratio of Cache Hits

The more frequently a get fails to find a valid value in the first cache and has to try a second cache, the more the overall performance is affected.

A common cause of misses is expiration or eviction of the entry. If you have a region's entry expiration or eviction enabled, monitor the region and entry statistics.

If you see a high ratio of misses to hits on the entries, consider increasing the expiration times or the maximum values for eviction, if possible. See [Eviction](#) on page 248 for more information.

## System Member Performance

You can modify some configuration parameters to improve system member performance.

Before doing so, you should understand [Basic Configuration and Programming](#) on page 131.

### Distributed System Member Properties

Several performance-related properties apply to a cache server or application that connects to the distributed system.

- **statistic-sampling-enabled.** Turning off statistics sampling 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.** Increasing 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 setting reduces system resource consumption. See [Logging](#) on page 543.

### JVM Memory Settings and System Performance

You configure JVM memory settings for the Java application by adding parameters to the java invocation. For the cache server, you add them to the command-line parameters for the `gfsh start server` command.

- JVM heap size—Your JVM may require more memory than is allocated by default. For example, you may need to increase heap size for an application 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 member, 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
```

Properties can be passed to the cache server on the `gfsh` command line:

```
gfsh>start server --name=server-name --J=-Xmx1024m --J=-Xms1024m
```

- MaxDirectMemorySize—The JVM has a kind of memory called direct memory, which is distinct from normal JVM heap memory, that can run out. You can increase the direct buffer memory either by increasing the maximum heap size (see previous JVM 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:

```
gfsh>start server --name=server-name --J=-XX:MaxDirectMemorySize=256M
```

- JVM stack size---Each thread in a Java application has its own stack. The stack is used to hold return addresses, arguments to functions and method calls, and so on. Since GemFire is a highly multi-threaded system, at any given point in time there are multiple thread pools and threads that are in use. The default stack size setting for a thread in Java is 1MB. Stack size has to be allocated in contiguous blocks and if the machine is being used actively and there are many threads running in the system (Task Manager shows the number of active threads), you may encounter an `OutOfMemory` error: `unable to create new native thread`, even though your process has enough available heap. If this happens, consider reducing the stack size requirement for threads on the cache server. The following parameter added to the Java application startup limits the maximum size of the stack.

```
-Xss384k
```

In particular, we recommend starting the cache servers with a stack size of 384k or 512k in such cases. For example:

```
gfsh>start server --name=server-name --J=-Xss384k
```

```
gfsh>start server --name=server-name --J=-Xss512k
```

### **Garbage Collection and System Performance**

If you experience unacceptably high latencies in application processing, you might improve performance by modifying your JVM's garbage collection behavior.

Garbage collection, while necessary, introduces latency into your system by consuming resources that would otherwise be available to your application.



**Note:** Garbage collection tuning options depend on the JVM you are using. Suggestions given here apply to the Sun HotSpot JVM. If you use a different JVM, check with your vendor to see if these or comparable options are available to you.



**Note:** 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 JVM 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 and Performance**

When many peer processes are started concurrently, you can improve the distributed system connect time 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
```

### **Slow Receivers with TCP/IP**

You have several options for preventing situations that can cause slow receivers of data distributions.

Before you begin, you should understand GemFire [Basic Configuration and Programming](#) on page 131.



**Note:** 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 Slow Receivers

During system integration, you can identify and eliminate potential causes of slow receivers in peer-to-peer communication.

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

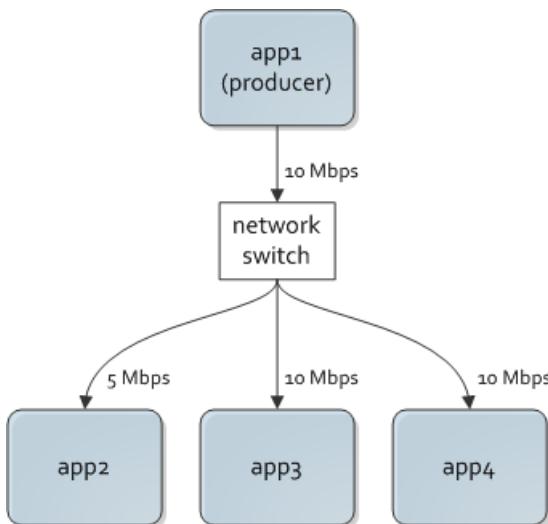
Make sure that the machines that run GemFire members have enough CPU available to them. Do not run any other heavyweight processes on the same machine.

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.



## 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

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 acknowledgments and distributed synchronization are required. 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 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.



**Note:** These slow receiver options are disabled in systems using SSL. See [SSL](#) on page 513.

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.

- **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.

- `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.

### 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.

### 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 exceed 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). See [Logging](#) on page 543.

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

These are the main use cases for the slow receiver specifications:

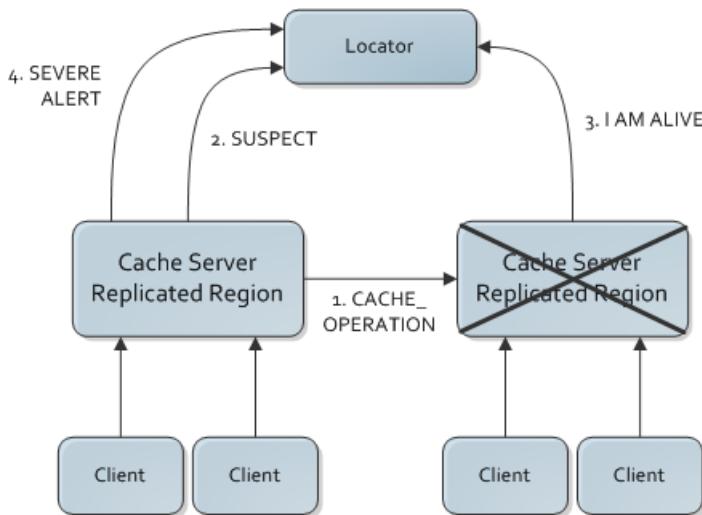
- Message bursts—With message bursts, the socket buffer can overflow and cause the producer to block. To keep from blocking, first make sure your socket buffer is large enough to handle a normal number of messages (using the `socket-buffer-size` property), then set the async distribution timeout to 1. 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 ("Membership: requesting removal of entry(#). Disconnected as a slow-receiver"). 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.



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.

```

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.



**Note:** 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.

## 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` 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 netSearches performed on regions of any distributed scope. (Note: `netSearch` is only performed on regions with a data-policy of empty, normal and preloaded.) 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)` method.
2. Reduce your buffer sizes to slow down the distributed-no-ack operations. 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.
  - 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 to something smaller than the default of 3.5 megabytes.
  - If you're using TCP/IP, reduce the `socket-buffer-size` in `gemfire.properties`.

## Socket Communication

GemFire processes communicate using TCP/IP and UDP unicast and multicast protocols. In all cases, communication uses sockets that you can tune to optimize performance.

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. Before you begin, you should understand GemFire [Basic Configuration and Programming](#) on page 131.

## Setting Socket Buffer Sizes

When you determine buffer size settings, you try 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.

Protocol / Area Affected	Configuration Location	Property Name
<b>TCP / IP</b>	---	---
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 sender	cache.xml <gateway-sender>	socket-buffer-size
Gateway receiver	cache.xml <gateway-receiver>	socket-buffer-size
<b>UDP Multicast</b>	---	---
Peer-to-peer send	gemfire.properties	mcast-send-buffer-size
Peer-to-peer receive	gemfire.properties	mcast-recv-buffer-size
<b>UDP Unicast</b>	---	---
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:

```
Client Socket Buffer Size cache.xml Configuration:
<pool>name="PoolA" server-group="dataSetA" socket-buffer-size="42000"...
```

```
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:

```
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>

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:

```
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

By default, Windows' ephemeral ports are within the range 1024-4999, inclusive. You can increase the range.

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. 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 available to your applications 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` 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.



**Note:** WAN deployments increase the messaging demands on a GemFire system. To avoid hangs related to WAN messaging, always set `conserve-sockets=false` for GemFire members that participate in a WAN deployment.

- **Client.** You can configure whether your clients share their socket connections to servers with the pool setting thread-local-connections. 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 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.

### 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 member'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 sender or gateway receiver
- 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 Socket Requirements Per Member

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.

In this table:

- 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.

Peer Member Socket Description	Number Used
Membership failure detection	2
Listener for incoming peer connections (server P2P)	1
Shared sockets (2 in and 2 out)	$4 * (M-1)$
Threads that share sockets use these.	

Peer Member Socket Description	Number Used
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 $(T*2)$



**Note:** 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.

### Server Socket Requirements Per 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 the server's 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 subscriptions.

In this table, M is the total number of members in the distributed system.

Server 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.	$(2 * \text{number of threads in a server that service client pool connections}) * (M-1)$ These threads do not share sockets.
Subscription connections	$2 * \text{number of client subscription connections to this server}$

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 as noted in the Peer-to-Peer section above.

### Client Socket Requirements per 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.

Client 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 as noted in the Peer-to-Peer section of this topic.

### Multi-site (WAN) Socket Requirements

Each gateway sender and gateway receiver uses a socket to distribute events or to listen for incoming connections from remote sites.

Multi-site Socket Description	Number Used
Listener for incoming connections	summation of the number of gateway-receivers defined for the member
Incoming connection	summation of the total number of remote gateway senders configured to connect to the gateway receiver
Outgoing connection	summation of the number of gateway senders defined for the member

Servers are peers in their own distributed system and have the additional socket requirements as noted in the Peer-to-Peer section above.

### TCP/IP Peer-to-Peer Handshake Timeouts

You can alleviate connection handshake timeouts for TCP/IP connections 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 gfsh command line:

```
gfsh>start server --name=server_name --J=-Dp2p.handshakeTimeoutMs=75000
```

## UDP Communication

You can make configuration adjustments to improve multicast and unicast UDP performance of peer-to-peer communication.

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 actions 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.

Before you begin, you should understand GemFire [Basic Configuration and Programming](#) on page 131. See also the general communication tuning and multicast-specific tuning covered in [Socket Communication](#) on page 530 and [Multicast Communication](#) on page 537.

## UDP Datagram Size

You can change the UDP datagram size with the GemFire property `udp-fragment-size`. 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 may 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 sender's 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`. This property is composed of:

- `byteAllowance`—Determines how many bytes (also referred to as credits) can be sent before receiving a recharge from the receiving processes.
- `rechargeThreshold`—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`—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. JVM 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.

## Multicast Communication

You can make configuration adjustments to improve the UDP multicast performance of peer-to-peer communication in your GemFire system.

Before you begin, you should understand GemFire [Basic Configuration and Programming](#) on page 131. See also the general communication tuning and UDP tuning covered in [Socket Communication](#) on page 530 and [UDP Communication](#) on page 536.

## Provisioning Bandwidth for Multicast

Multicast installations require more planning and configuration than TCP installations. With 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.

## 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 must determine and explicitly set those limits.

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



**Note:** 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



**Note:** 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.

#### Output From Iperf Testing:

```
[ ID] Interval Transfer Bandwidth Jitter Lost/Total Datagrams
[  3] 0.0- 1.0 sec  129 KBytes  1.0 Mbits/sec 0.778 ms  61/ 151 (40%)
[  3] 1.0- 2.0 sec  128 KBytes  1.0 Mbits/sec 0.236 ms   0/   89 (0%)
[  3] 2.0- 3.0 sec  128 KBytes  1.0 Mbits/sec 0.264 ms   0/   89 (0%)
```

```
[  3] 3.0- 4.0 sec 128 KBytes 1.0 Mbits/sec 0.248 ms 0/ 89 (0%)
[  3] 0.0- 4.3 sec 554 KBytes 1.0 Mbits/sec 0.298 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

After you determine the maximum transmission rate, 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 on the gfsh the command line:

```
gfsh>start server --name=server_name --enable-time-statistics
```

- 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. If the cache regions are configured to require acknowledgment, 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.
- Slow system performance might be helped by reducing how far your multicast messaging goes in your network.
- 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 when starting a cache server process through the gfsh command line:

```
gfsh>start server --name=server_name --J=-Dp2p.disableBatching=true
```

### Run-time Considerations for Multicast

When you use multicast for messaging and data distribution, you need to understand how the health monitoring setting works and how to control memory use.

## 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
```



**Note:** 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

Several problems may arise 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.

## Maintaining Cache Consistency

Maintaining data consistency between caches in a distributed GemFire system is vital for ensuring its functional integrity and preventing data loss.

### Before Restarting a Region with a Disk Store, Consider the State of the Entire Region



**Note:** If you revoke a member's disk store, do not restart that member with its disk stores—in isolation—at a later time.

GemFire stores information about your persisted data and prevents you from starting a member with a revoked disk store in the running system. But GemFire cannot stop you from starting a revoked member in isolation, and running with its revoked data. This is an unlikely situation, but it is possible to do:

1. Members A and B are running, both storing Region data to disk.
2. Member A goes down.
3. Member B goes down.
4. At this point, Member B has the most recent disk data.
5. Member B is not usable. Perhaps its host machine is down or cut off temporarily.
6. To get the system up and running, you start Member A, and use the command line tool to revoke Member B's status as member with the most recent data. The system loads Member A's data and you run forward with that.
7. Member A is stopped.
8. At this point, both Member A and Member B have information in their disk files indicating they are the gold copy members.
9. If you start Member B, it will load its data from disk.
10. When you start Member A, the system will recognize the incompatible state and report an exception, but by this point, you have good data in both files, with no way to combine them.

### Prevent Primary and Secondary Gateway Senders from Going Offline

In a multi-site installation, if the primary gateway server goes offline, a secondary gateway sender must take over primary responsibilities as the failover system. The existing secondary gateway sender detects that the primary gateway sender has gone offline, and a secondary one becomes the new primary. Because the queue is distributed, its contents are available to all gateway senders. So, when a secondary gateway sender becomes primary, it is able to start processing the queue where the previous primary left off with no loss of data.

If both the primary gateway sender and all its secondary senders go offline and messages are in their queues, data loss could occur, because there is no failover system.

### Avoid Using the Admin Interface to Start a Server

Avoid using the GemFire admin interface to start a server and specify the ping interval. When you specify the ping interval, it is not sent to the new server, so this could cause the server to adopt a zero-millisecond interval. Consequently, the server terminates all client connections. The symptoms of this behavior are clients that continually move their interest lists between servers, clients that do not receive cache update notifications, or only receive them sporadically.

### Verify That `isOriginRemote` Is Set to False

The `isOriginRemote` flag for a server or a multi-site gateway is set to false by default, which ensures that updates are distributed to other members. Setting its value to true in the server or the receiving gateway member applies updates to that member only, so updates are not distributed to peer members.

### Optimize `socket-buffer-size`

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 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. You can configure the send-receive buffer sizes of the TCP/IP connections used for data transmissions by changing the socket-buffer-size attribute of the gateway-sender and gateway-receiver elements in the `cache.xml` file. Set the buffer size by determining the link bandwidth and then using ping to measure the round-trip time. Use the same value for both gateway senders and gateway receivers.

### **Understand Cache Transactions**

Understanding the operation of GemFire transactions can help you minimize situations where the cache could get out of sync.

Transactions do not work in distributed regions with global scope.

Transactions provide consistency within one cache, but the distribution of results to other members is not as consistent.

Multiple transactions in a cache can create inconsistencies because of read committed isolation. Since multiple threads cannot participate in a transaction, most applications will be running multiple transactions.

An in-place change to directly alter a key's value without doing a put can result in cache inconsistencies. With transactions, it creates additional difficulties because it breaks read committed isolation. If at all possible, use copy-on-read instead.

In distributed-no-ack scope, two conflicting transactions in different members can commit simultaneously, overwriting each other as the changes are distributed.

If a cache writer exists during a transaction, then each transaction write operation triggers a cache writer's related call. Regardless of the region's scope, a transaction commit can invoke a cache writer only in the local cache and not in the remote caches.

A region in a cache with transactions may not stay in sync with a region of the same name in another cache without transactions.

Two applications running the same sequence of operations in their transactions may get different results. This could occur because operations happening outside a transaction in one of the members can overwrite the transaction, even in the process of committing. This could also occur if the results of a large transaction exceed the machine's memory or the capacity of GemFire. Those limits can vary by machine, so the two members may not be in sync.

## **Logging**

Comprehensive logging messages help you confirm system configuration and debug problems in configuration and code.

### **Categories of Log Messages**

System logging messages typically pertain to startup; logging management; connection and system membership; distribution; or cache, region, and entry management.

- **Startup information.** Describe the Java version, the GemFire native version, the host system, current working directory, and environment settings. These messages contain all information about the system and configuration the process is running with.
- **Logging management.** 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).
- **Connections and system membership.** 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.** Report on the distribution of data between system members. These messages include information about region configuration, entry creation and modification, and region and entry invalidation and destruction.
- **Cache, region, and entry management.** Cache initialization, listener activity, locking and unlocking, region initialization, and entry updates.

## Structure of a Log Message

Every logged message contains:

- The message header within square brackets:
  1. The message level
  2. The time the message was logged
  3. The ID of the connection and thread that logged the message, which might be the main program or a system management process
- The message itself, which can be a string and/or an exception with the exception stack trace

```
[config 2005/11/08 15:46:08.710 PST PushConsumer main nid=0x1]
Cache initialized using "file:/Samples/quickstart/xml/PushConsumer.xml".
```

## Log File Name

Specify your GemFire system member's main log in the `gemfire` property `log-file` setting.

GemFire uses this name for the most recent log file, actively in use if the member is running, or used for the last run. GemFire creates the main log file when the application starts.

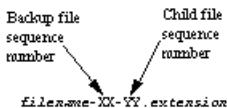
By default, the main log contains the entire log for the member session. If you specify a `log-file-size-limit`, GemFire splits the logging into these files:

- **The main, current log.** Holding current logging entries. Named with the string you specified in `log-file`.
- **Child logs.** Holding older logging entries. These are created by renaming the main, current log when it reaches the size limit.
- **A metadata log file, with `meta-` prefixed to the name.** Used to track of startup, shutdown, child log management, and other logging management operations

The current log is renamed, or rolled, to the next available child log when the specified size limit is reached.

When your application connects with logging enabled, it creates the main log file and, if required, the `meta-log` file. If the main log file is present when the member starts up, it is renamed to the next available child log to make way for new logging.

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 the `log-file` specification



If child logs are not used, the child file sequence number is a constant 00 (two zeros).

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 log 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`.

To figure out the member's most recent activities, look at the `meta-` log file or, if no meta file exists, the main log file.

## How the System Renames 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 `log-file=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 Nov 3 11:07 statArchive-01-00.gfs
-rw-rw-r-- 1 jpearson users 56556 Nov 3 11:08 statArchive-02-00.gfs
-rw-rw-r-- 1 jpearson users 56965 Nov 3 11:09 statArchive-03-00.gfs
-rw-rw-r-- 1 jpearson users 11308 Nov 3 11:27 system-01-00.log
-rw-rw-r-- 1 jpearson users 59650 Nov 3 11:34 statArchive.gfs
-rw-rw-r-- 1 jpearson users 18178 Nov 3 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 Nov 3 11:07 statArchive-01-00.gfs
-rw-rw-r-- 1 jpearson users 56556 Nov 3 11:08 statArchive-02-00.gfs
-rw-rw-r-- 1 jpearson users 56965 Nov 3 11:09 statArchive-03-00.gfs
-rw-rw-r-- 1 jpearson users 11308 Nov 3 11:27 system-01-00.log
-rw-rw-r-- 1 jpearson users 59650 Nov 3 11:34 statArchive-04-00.gfs
-rw-rw-r-- 1 jpearson users 18178 Nov 3 11:34 system-04-00.log
-rw-rw-r-- 1 jpearson users 55774 Nov 4 10:08 statArchive.gfs
-rw-rw-r-- 1 jpearson users 17681 Nov 4 10:08 system.log
```

The statistics and the log file are renamed using the next integer that is available to both, so the log file sequence jumps past the gap in this case.

## Log Level

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, with sample output:

- **severe (highest level)**. This level indicates a serious failure. In general, severe messages describe events that are of considerable importance that 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 noted in the error message failed.

This error was produced by throwing a `Throwable` from a `CacheListener`. While dispatching events to a customer-implemented cache listener, 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
```

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 is for informational messages, typically geared to end users and system administrators.

This is a typical info message created at system member startup. This indicates that no other `DistributionManagers` are running in the distributed system, which means no other system members are running:

```
[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.

You can use this config message 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"
...
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.



**Note:** Generally, you should only use this level if instructed to do so by VMware 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 2011/06/21 11:27:24.689 PDT <locatoragent_ds_w1-gst-dev04_2104>
tid=0xe] SSL Configuration:
    ssl-enabled = false
```

- **finer, finest, and all.** These levels exist for internal use only. They produce a large amount of data and so consume large amounts of disk space and system resources.



**Note:** Do not use these settings unless asked to do so by VMware technical support.

## Naming, Searching, and Creating Log Files

The best way to manage and understand the logs is to have each member log to its own files.

## Log File Naming Recommendation

For members running on the same machine, you can have them log to their own files by starting them in different working directories and using the same, relative log-file specification. For example, you could set this in <commonDirectoryPath>/gemfire.properties:

```
log-file=../log/member.log
```

then start each member in a different directory with this command, which points to the common properties file:

```
java -DgemfirePropertyFile=<commonDirectoryPath>/gemfire.properties
```

This way, each member has its own log files under its own working directory.

## Searching the Log Files

For the clearest picture, merge the log files, with the gfsh export logs command:

```
gfsh> export logs --dir=myDir --dir=myDir --merge-log=true
```

Search for lines that begin with these strings:

- [warning]
- [error]
- [severe]

## Creating Your Own Log Messages

In addition to the system logs, 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 your logging configuration settings.

## Set Up Logging

You configure logging in a member's gemfire.properties or at startup with gfsh.

Before you begin, make sure you understand [Basic Configuration and Programming](#) on page 131.

1. Run a time synchronization service such as NTP on all GemFire host machines. This is the only way to produce logs that are useful for troubleshooting. Synchronized time stamps ensure that log messages from different hosts can be merged to accurately reproduce a chronological history of a distributed run.
2. Use a sniffer to monitor your logs and, if you begin seeing new or unexpected warnings, errors, or severe messages, contact VMware technical support. The logs output by your system have their own characteristics, indicative of your system configuration and of the particular behavior of your applications, so you must become familiar with your applications' logs to use them effectively.
3. Configure member logging in each member's gemfire.properties as needed:

```
# Default gemfire.properties log file settings
log-level=config
log-file=
log-file-size-limit=0
log-disk-space-limit=0
```



**Note:** You can also specify logging parameters when you start up members (either locators or servers) using the gfsh command-line utility. See [start](#) on page 713 commands for various examples. In addition, you can modify log file properties and log-level settings while a member is already running by using the [alter runtime](#) on page 674 command.

- a. Set log-level. Options are **severe** (the highest level), **error**, **warning**, **info**, **config**, and **fine**. The lower levels include higher level settings, so a setting of **warning** would log **warning**,

error, and severe messages. For general troubleshooting, we recommend setting the log level at config or higher.

 **Note:** The fine setting can fill up disk too quickly and impact system performance. Use fine only if asked to do so by VMware technical support.

- b. Specify the log file name in `log-file`. This can be relative or absolute. If this property is not specified, the defaults are:
  - Standard output for applications
  - For cache servers, the default log file location is:

```
working-directory/server-name.log
```

By default, when starting a server through `gfsh`, the `working -directory` corresponds to the directory (named after itself) that the cache server creates upon startup. Alternatively, you can specify a different working directory path when you start the cache server. The `server-name` corresponds to the name of the cache server provided upon startup.

- For a standalone locator, the default log file location is:

```
working-directory/locator-name.log
```

By default, when starting a locator through `gfsh`, the `working -directory` corresponds to the directory (named after itself) created when the locator starts up. Alternatively, you can specify a different working directory path when you start a locator. The `locator-name` corresponds to the name of the locator provided upon startup. If you are using a colocated or embedded locator, the locator logs will be part of the member's log file.

For the easiest logs examination and troubleshooting, send your logs to files instead of standard out.

 **Note:** Make sure each member logs to its own files. This makes the logs easier to decipher.

- c. Set the maximum size of a single log file in `log-file-size-limit`. If not set, the single, main log file is used. If set, the metadata file, the main log, and rolled child logs are used.
- d. Set the maximum size of all log files in `log-disk-space-limit`. If non-zero, this limits the combined size of all inactive log files, deleting oldest files first to stay under the limit. A zero setting indicates no limit.
4. If you are using the `gfsh` command-line interface, `gfsh` creates its own log file in the directory where you run the `gfsh` or `gfsh.bat` script. By default, these log files are named `gfsh-YYYY-MM-DD_HH-MM-SS.log` where `YYYY-MM-DD_HH-MM-SS` corresponds to the local date and timestamp of when you started the `gfsh` terminal session.



# Chapter 46

---

## Statistics

Every application and server in a distributed system can access statistical data about vFabric GemFire operations. You can configure this data in the `gemfire.properties` file or in `gfsh` to facilitate collection and analysis.

### How Statistics Work

Each application or cache server that joins the distributed system can collect and archive statistical data for analyzing system performance.

Set the configuration attributes that control statistics collection in the `gemfire.properties` configuration file. You can also collect your own application defined statistics. To view and analyze archived historical data, use the Visual Statistics Display (VSD) utility provided in a separate download from the product. Contact VMware technical support for instructions about acquiring VSD.

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.



**Note:** GemFire statistics use the Java `System.nanoTime` for nanosecond timing. This method provides nanosecond precision, but not necessarily nanosecond accuracy. For more information, see the online Java documentation for `System.nanoTime` for the JRE you are using with GemFire.

Statistics sampling provides valuable information for ongoing system tuning and troubleshooting. Sampling statistics (not including time-based statistics) at the default sample rate does not impact overall distributed system performance. We recommend enabling statistics sampling in production environments. We do not recommend enabling time-based statistics (configured with the `enable-time-statistics` property) in production.

### Transient Region and Entry Statistics

For replicated, distributed, and local regions, GemFire provides a standard set of statistics for the region and its entries.

GemFire gathers these statistics when the region attribute `statistics-enabled` is set to true.



**Note:** Unlike other GemFire statistics, these region and entry statistics are not archived and cannot be charted.



**Note:** Enabling these statistics requires extra memory per entry. See [Memory Requirements for Cached Data](#) on page 989.

These are the transient statistics gathered for all but partitioned regions:

- **Hit and miss counts.** For the entry, the hit count is the number of times the cached entry was accessed through the `Region.get` method and the miss count is the number of times these hits did not find a valid value. For the region these counts are the totals for all entries in the region. The API provides `get` methods for the hit and miss counts, a convenience method that returns the hit-to-miss ratio, and a method for zeroing the counts.
- **Last accessed time.** For the entry, this is the last time a valid value was retrieved from the locally cached entry. For the region, this is the most recent "last accessed time" for all entries contained in the region. This statistic is used for idle timeout expiration activities.
- **Last modified time.** For the entry, this is the last time the entry value was updated (directly or through distribution) due to a load, create, or put operation. For the region, this is the most recent "last modified time" for all entries contained in the region. This statistic is used for time to live and idle timeout expiration activities.

The hit and miss counts collected in these statistics can be useful for fine-tuning your system's caches. If you have a region's entry expiration enabled, for example, and see a high ratio of misses to hits on the entries, you might choose to increase the expiration times.

Retrieve region and entry statistics through the `getStatistics` methods of the `Region` and `Region.Entry` objects.

## Application-Defined and Custom Statistics

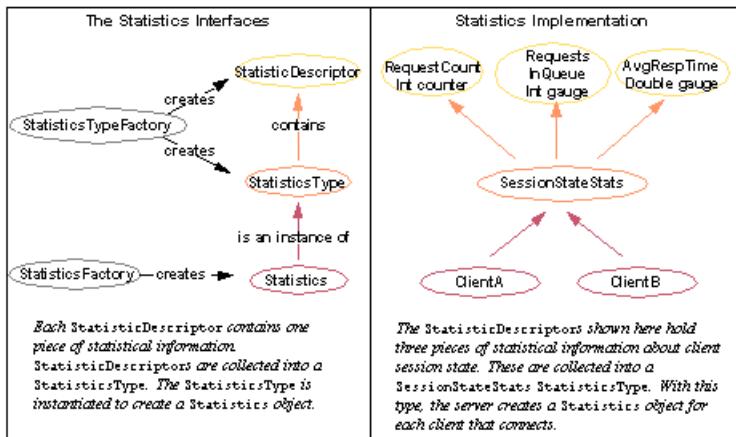
GemFire includes interfaces for defining and maintaining your own statistics.

The GemFire package, `com.gemstone.gemfire`, includes the following interfaces for defining and maintaining your own statistics:

- **StatisticDescriptor.** Describes an individual statistic. Each statistic has a name and information on the statistic it holds, such as its class type (long, int, etc.) and whether it is a counter that always increments, or a gauge that can vary in any manner.
- **StatisticsType.** Logical type that holds a list of `StatisticDescriptor`s and provides access methods to them. The `StatisticDescriptor`s contained by a `StatisticsType` are each assigned a unique ID within the list. `StatisticsType` is used to create a `Statistics` instance.
- **Statistics.** Instantiation of an existing `StatisticsType` object with methods for setting, incrementing, and getting individual `StatisticDescriptor` values.
- **StatisticsFactory.** Creates instances of `Statistics`. You can also use it to create instances of `StatisticDescriptor` and `StatisticsType`, because it implements `StatisticsTypeFactory`. `DistributedSystem` is an instance of `StatisticsFactory`.
- **StatisticsTypeFactory.** Creates instances of `StatisticDescriptor` and `StatisticsType`.

The statistics interfaces are instantiated using statistics factory methods that are included in the package. For coding examples, see the online Java API documentation for `StatisticsFactory` and `StatisticsTypeFactory`.

As an example, an application server might collect statistics on each client session in order to gauge whether client requests are being processed in a satisfactory manner. Long request queues or long server response times could prompt some capacity-management action such as starting additional application servers. To set this up, each session-state data point is identified and defined in a `StatisticDescriptor` instance. One instance might be a `RequestsInQueue` gauge, a non-negative integer that increments and decrements. Another could be a `RequestCount` counter, an integer that always increments. A list of these descriptors is used to instantiate a `SessionStateStats` `StatisticsType`. When a client connects, the application server uses the `StatisticsType` object to create a session-specific `Statistics` object. The server then uses the `Statistics` methods to modify and retrieve the client's statistics. This figure illustrates the relationships between the statistics interfaces and shows the implementation of this use case.



## Configuring and Using Statistics

You configure statistics and statistics archiving in gemfire.properties

- [Configure Statistics](#) on page 553
  - [Controlling the Size of Archive Files](#) on page 554
- [page 555](#)

### Configure Statistics

In this procedure it is assumed that you understand [Basic Configuration and Programming](#) on page 131.

1. Configure gemfire.properties for the statistics monitoring and archival that you need:
  - a. Enable statistics gathering for the distributed system. This is required for all other statistics activities:
 

```
statistic-sampling-enabled=true
```

 **Note:** Statistics sampling at the default sample rate (1000 milliseconds) does not impact system performance and is recommended in production environments for troubleshooting.
  - b. Change the statistics sample rate as needed. Example:
 

```
statistic-sampling-enabled=true
statistic-sample-rate=2000
```
  - c. To archive the statistics to disk, enable that and set any file or disk space limits that you need. Example:
 

```
statistic-sampling-enabled=true
statistic-archive-file=myStatisticsArchiveFile.gfs
archive-file-size-limit=100
archive-disk-space-limit=1000
```
  - d. If you need time-based statistics, enable that. Time-based statistics require statistics sampling and archival. Example:
 

```
statistic-sampling-enabled=true
statistic-archive-file=myStatisticsArchiveFile.gfs
enable-time-statistics=true
```

 **Note:** Time-based statistics can impact system performance and is not recommended for production environments.

2. Enable transient region and entry statistics gathering on the regions where you need it. Expiration requires statistics. Example:

```
<region name="myRegion" refid="REPLICATE">
  <region-attributes statistics-enabled="true">
  </region-attributes>
</region>
```



**Note:** Region and entry statistics are not archived and can only be accessed through the API. As needed, retrieve region and entry statistics through the `getStatistics` methods of the `Region` and `Region.Entry` objects. Example:

```
out.println("Current Region:\n\t" + this.currRegion.getName());
RegionAttributes attrs = this.currRegion.getAttributes();
if (attrs.getStatisticsEnabled()) {
    CacheStatistics stats = this.currRegion.getStatistics();
    out.println("Stats:\n\tHitCount is " + stats.getHitCount() +
    "\n\tMissCount is " + stats.getMissCount() +
    "\n\tLastAccessedTime is " + stats.getLastAccessedTime() +
    "\n\tLastModifiedTime is " + stats.getLastModifiedTime());
}
```

3. Create and manage any custom statistics that you need through the `cache.xml` and the API. Example:

```
// Create custom statistics
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE statistics PUBLIC
"-//GemStone Systems, Inc.//GemFire Statistics Type//EN"
"http://www.gemstone.com/dtd/statisticsType.dtd">
<statistics>
  <type name="StatSampler">
    <description>Stats on the statistic sampler.</description>
    <stat name="sampleCount" storage="int" counter="true">
      <description>Total number of samples taken by this
sampler.</description>
      <unit>samples</unit>
    </stat>
    <stat name="sampleTime" storage="long" counter="true">
      <description>Total amount of time spent taking
samples.</description>
      <unit>milliseconds</unit>
    </stat>
  </type>
</statistics>

// Update custom stats through the API
this.samplerStats.incInt(this.sampleCountId, 1);
this.samplerStats.incLong(this.sampleTimeId, nanosSpentWorking / 1000000);
```

4. Access archived statistics through these utilities:

- The `gfsh show metrics` command. See [show metrics](#) on page 710 for more information.
- VSD, the Visual Statistics Display tool. See [Visual Statistics Display](#) on page 839 for more information.

## 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 `gemfire.properties statistic-sample-rate` controls how often samples are taken, which affects the speed at which the archive file grows.
- The `gemfire.properties statistic-archive-file` controls whether the statistics files are compressed. If you give the file name a `.gz` suffix, it is compressed, thereby taking up less disk space.
- **Maximum Size of a Single Archive File.** If the value of the `gemfire.properties archive-file-size-limit` is greater than zero, a new archive is started when the size of the current archive exceeds the limit. Only one archive can be active at a time.



**Note:** 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 `gemfire.properties archive-disk-space-limit` controls the maximum size of all inactive archive files combined. By default, the limit is set to 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 the 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.



**Note:** 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.

## Examining Archived Statistics

When sampling and archiving are enabled, you can study statistics in archive files through VSD or by using the `gfsh show metrics` 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.

## GemFire Statistics List

This section describes the primary statistics gathered by GemFire when statistics are enabled.

All statistics gathering requires the `gemfire.properties statistic-sampling-enabled` in the `gemfire.properties` file to be true. Statistics that use time require the `gemfire.properties enable-time-statistics` to be true.

## System Performance Statistics

Performance statistics are collected for each Java application or cache server that connects to a distributed system.



**Note:** For pure Java applications, operating system statistics are not available for collection. Operating system statistics are categorized as process statistics and system statistics.

## CacheClientNotifierStatistics

Statistics regarding cache server operations sent to clients.

Statistic	Description
events	Number of events (operations) processed by the cache client notifier.

Statistic	Description
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.
eventsEnqueuedWhileClientAwayCount	Number of events enqueued for a durable client.
cqProcessingTime	Total time, in nanoseconds, spent by the cache client notifier processing CQs.

### CacheClientProxyStatistics

Statistics regarding cache server operations and cache server client notifications sent to a single client.

Statistic	Description
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.

## CacheClientUpdaterStats

Statistics in a client that 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). :

Statistic	Description
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.

## CachePerfStats

Statistics on the GemFire cache. These can be used to determine the type and number of cache operations being performed and how much time they consume. The primary statistics are:

Statistic	Description
conflatedEvents	The number of events that were conflated, and not delivered to event listeners or gateway senders on this member. Events are typically conflated because a later event was already applied to the cache, or because a concurrent event was ignored to ensure cache consistency. Note that some members may discard an update while other members apply the update, depending on the order in which each member receives the update. For this reason, the <code>conflatedEvents</code> statistic will differ for each GemFire member. See <a href="#">How Consistency Checking Works for Replicated Regions</a> on page 241.
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 <code>get()</code> 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 <code>get()</code> 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 <code>get()</code> 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.

Statistic	Description
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 gett() 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.

Statistic	Description
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.
tombstoneCount	The total number of tombstone entries created for performing concurrency checks. See <a href="#">How Destroy and Clear Operations Are Resolved</a> on page 243.
tombstoneGCCount	The total number of tombstone garbage collection cycles that a member has performed. See <a href="#">How Destroy and Clear Operations Are Resolved</a> on page 243.
replicatedTombstonesSize	The approximate number of bytes that are currently consumed by tombstones in replicated or partitioned regions. See <a href="#">How Destroy and Clear Operations Are Resolved</a> on page 243.
nonReplicatedTombstonesSize	The approximate number of bytes that are currently consumed by tombstones in non-replicated regions. See <a href="#">How Destroy and Clear Operations Are Resolved</a> on page 243.

## CacheServerStats

Statistics used for cache servers and for gateway receivers and are recorded in CacheServerStats in a cache server. The primary statistics are:

Statistic	Description
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.

Statistic	Description
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.
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.

Statistic	Description
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.

Statistic	Description
loadPerConnection	The estimate of how much load is 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

## ClientStats

These statistics are in a client and they describe all the messages sent from the client to a specific server. The primary statistics are:

Statistic	Description
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.

Statistic	Description
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.

Statistic	Description
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.

Statistic	Description
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.

Statistic	Description
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.

Statistic	Description
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.

Statistic	Description
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.

Statistic	Description
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.

### ClientSubscriptionStats

Collected in the server, these 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. The primary statistics are:

Statistic	Description
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.

## CQStatistics

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:

Statistic	Description
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.

## DeltaPropagationStatistics

These statistics are for delta propagation between members. The primary statistics are:

Statistic	Description
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 extracting deltas.
partitionMessagesWithDeltaProcessed	Number of PartitionMessages containing delta processed.
partitionMessagesWithDeltaProcessedTime	Total time spent applying deltas.

Statistic	Description
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 successfully.
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:

Statistic	Description
diskSpace	The total number of bytes currently being used on disk in this directory.

### DiskRegionStatistics

Statistics regarding operations performed on a disk region for persistence/overflow. The primary statistics are:

Statistic	Description
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 member. The value may also have been written to the disk.

## DistributionStats

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.

The primary statistics are:

Statistic	Description
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.

Statistic	Description
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.

Statistic	Description
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 members 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.

Statistic	Description
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.
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.

Statistic	Description
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 datograms.
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 datograms.
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.

Statistic	Description
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.

Statistic	Description
jgSTABILITYmessages	Number of STABILITY messages received by JGroups.
jgUDPupTime	Time, in nanoseconds, 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.

Statistic	Description
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.

Statistic	Description
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.
pdxSerializations	Total number of PDX serializations.
pdxSerializedBytes	Total number of bytes produced by PDX serialization.
pdxDeserializations	Total number of PDX deserializations.
pdxDeserializedBytes	Total number of bytes read by PDX deserialization.
pdxInstanceDeserializations	Total number of times getObject has been called on a PdxInstance.
pdxInstanceDeserializationTime	Total amount of time, in nanoseconds, spent deserializing PdxInstances by calling getObject.
pdxInstanceCreations	Total number of times a PdxInstance has been created by deserialization.

### DistributionStats Related to Slow Receivers

The distribution statistics provide statistics pertaining to slow receivers. The primary statistics are:

Statistic	Description
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.

Statistic	Description
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.

## DLockStats

These statistics are for distributed lock services. The primary statistics are:

Statistic	Description
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.

Statistic	Description
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).

Statistic	Description
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.
destroyReadWaitsInProgressDesc	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.

## FunctionServiceStatistics

These statistics are the aggregate statistics for all function executions. The primary statistics are:

Statistic	Description
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.

## FunctionStatistics

These are the statistics for each execution of the function. The primary statistics are:

Statistic	Description
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.

## GatewayStatistics

These statistics are for an outgoing gateway queue and connection. The primary statistics are:

Statistic	Description
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.

Statistic	Description
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.

### GatewayHubStatistics

These statistics are for the WAN gateway hub. The primary statistics are:

Statistic	Description
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.

## LocatorStatistics

These statistics are on the GemFire locator. The primary statistics are:

Statistic	Description
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.

## LRUStatistics – Count-based

The entry-count least recently used (LRU) eviction mechanism records these LRUStatistics. The primary statistics are:

Statistic	Description
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.

## LRUStatistics – 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:

Statistic	Description
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.

Statistic	Description
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.

### PartitionedRegion<partitioned\_region\_name>Statistics

#### Partitioned Region 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.



**Note:** Unsuccessful operations are not counted in these statistics.

The primary statistics are:

Statistic	Description
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.

Statistic	Description
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.

### Partitioned Region Statistics on Partition Messages



**Note:** Unsuccessful operations and local operations—those that originated in this member—are not counted in these statistics.

The primary statistics are:

Statistic	Description
partitionMessagesProcessed	Number of region operations executed in this member at the request of other data stores 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.

### Partitioned Region Statistics on Data Entry Caching

These statistics track the pattern of data entry distribution among the buckets in this member. The primary statistics are:

Statistic	Description
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 member.
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 member.
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.

### Partitioned Region Statistics on Redundancy

These statistics track status on partitioned region data copies. The primary statistics are:

Statistic	Description
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.

Statistic	Description
actualRedundantCopies	<p>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 data store is lost or rise when a data store is added. This value may drop temporarily during partitioned region creation or destruction and then rise again.</p> <p> <b>Note:</b> If this value remains low, then partitioned region data is at risk and may be lost if another data store is lost.</p> <p>A healthy partitioned region will maintain a value equal to configuredRedundantCopies. The user should add one or more data stores 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 data stores present (that is, killing one data store will cause its data to fail over to another data store).</p>
lowRedundancyBucketCount	<p>The number of buckets in this partitioned region that currently have fewer copies than the configuredRedundantCopies. This value may rise above zero when a data store is lost and return to zero when one or more data stores are added. This value may rise temporarily during partitioned region creation or destruction and then return to zero.</p> <p> <b>Note:</b> If this value remains above zero, then partitioned region data is at risk and may be lost if another data store is lost.</p> <p>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 data stores present (that is, killing one data store will cause its data to fail over to another data store).</p>

## PoolStats

These statistics are in a client and they describe one of the client's connection pools. The primary statistics are:

Statistic	Description
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.

Statistic	Description
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.

### LinuxProcessStats

Operating system statistics on the member'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 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:

Statistic	Description
imageSize	Size, in megabytes, of the process's image.
rssSize	Size, in megabytes, of the process's resident size.

### SolarisProcessStats

Operating system statistics on the member 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 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:

Statistic	Description
allOtherSleepTime	The number of milliseconds the process has been sleeping for some reason not tracked by any other stat. Note, all lightweight 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.

Statistic	Description
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 of 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.

Statistic	Description
cpuUsed	The percentage of recent CPU time used by the process.
memoryUsed	The percentage of real memory used by the process.

### WindowsProcessStats

Operating system statistics on the member 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 runs without the use of the GemFire native library. These are the equivalent of SolarisProcessStats when running on Windows. The primary statistics are:

Statistic	Description
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.

Statistic	Description
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.

## ResourceManagerStats

Statistics related to the GemFire resource manager. Use these to help analyze and tune your JVM memory settings and the GemFire resource-manager settings. The primary statistics are:

Statistic	Description
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. The primary statistics are:

Statistic	Description
sampleCount	Total number of samples taken by this sampler.
sampleTime	Total amount of time spent taking samples.

## LinuxSystemStats

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:

Statistic	Description
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.

Statistic	Description
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 loopback).
recvBytes	Total number of network bytes received (excluding loopback).
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 loopback).
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.

Statistic	Description
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.

### SolarisSystemStats

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:

Statistic	Description
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.

Statistic	Description
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.

<b>Statistic</b>	<b>Description</b>
<code>invlContextSwitches</code>	The total number of times a thread was forced to give up the CPU even though it was still ready to run.
<code>majorPageFaults</code>	The total number of times a page fault required disk IO to get the page.
<code>messageCount</code>	The total number of msgrcv and msgsnd system calls.
<code>pageDaemonCycles</code>	The total number of revolutions of the page daemon's scan "clock hand".
<code>pageIns</code>	The total number of times pages have been brought into memory from disk by the operating system's memory manager.
<code>pageOuts</code>	The total number of times pages have been flushed from memory to disk by the operating system's memory manager.
<code>pagerRuns</code>	The total number of times the pager daemon has been scheduled to run.
<code>pagesPagedIn</code>	The total number of pages that have been brought into memory from disk by the operating system's memory manager.
<code>pagesPagedOut</code>	The total number of pages that have been flushed from memory to disk by the operating system's memory manager.
<code>pagesScanned</code>	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.
<code>procsInIoWait</code>	The number of processes waiting for block I/O at this instant in time.
<code>protectionFaults</code>	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.
<code>semaphoreOps</code>	The total number of semaphore operations.
<code>softwareLockFaults</code>	The total number of fault operations caused by software locks held on memory pages.
<code>systemCalls</code>	The total number of fault operations caused by software locks held on memory pages.
<code>systemMinorFaults</code>	The total number of minor page fault operations in kernel code. Minor page faults do not require disk access.
<code>threadCreates</code>	The total number of times a thread operation has been created.
<code>traps</code>	The total number of trap operations that have occurred on the computer.
<code>userMinorFaults</code>	The total number of minor page fault operations in non-kernel code. Minor page faults do not require disk access.
<code>loopbackInputPackets</code>	The total number of input packets received over the loopback network adaptor.

Statistic	Description
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	Octets 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.

## VMStats

Show the JVM'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:

Statistic	Description
cpus	Number of CPUs available to the member on its machine.
daemonThreads	Current number of live daemon threads in this JVM.
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 JVM started.
maxMemory	The maximum amount of memory, measured in bytes, that the JVM will attempt to use.
peakThreads	High water mark of live threads in this JVM.

Statistic	Description
pendingFinalization	Number of objects that are pending finalization in the JVM.
processCpuTime	CPU time, measured in nanoseconds, used by the process.
threads	Current number of live threads (both daemon and non-daemon) in this JVM.
threadStarts	Total number of times a thread has been started since this JVM 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 JVM started.

### VMGCStats

These statistics show how much time used by different JVM garbage collection and are available on JDK 1.5 and later JVMs. The primary statistics are:

Statistic	Description
collections	Total number of collections this garbage collector has done.
collectionTime	Approximate elapsed time spent doing collections by this garbage collector.

### VMMemoryPoolStats

These statistics describe memory usage in difference garbage collector memory pools. The primary statistics are:

Statistic	Description
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 JVM 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.

## VMMemoryUsageStats

Show details on how the Java heap memory is being used. This statistic is available on JDK 1.5 and later JVMs. The primary statistics are:

Statistic	Description
committedMemory	The amount of committed memory, measured in bytes, for this area.
initMemory	Initial memory the JVM 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.

## 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. The primary statistics are:

Statistic	Description
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.
txConflictCheckTime	The total amount of time, in nanoseconds, spent doing conflict checks during transaction commit.

### **IndexStats - Query Independent Index Statistics**

<b>Statistic</b>	<b>Description</b>
numKeys	Number of keys currently stored in the Index.
numValues	Number of values currently stored in the Index.
numUpdates	Number of updates applied and completed on the Index while inserting, modifying , or deleting corresponding data in GemFire.
updateTime	Total time taken in applying and completing updates on the Index.
updatesInProgress	Current number of updates in progress on the Index. Concurrent updates on an index are allowed.

### **IndexStats - Query Dependent Index Statistics**

<b>Statistic</b>	<b>Description</b>
numUses	Number of times the Index has been used for querying.
useTime	Total time during the use of the Index for querying.
usesInProgress	Current number of uses of the index in progress or current number of concurrent threads accessing the index for querying. Concurrent use of an index is allowed for different queries.

# Chapter 47

---

## Troubleshooting and System Recovery

This section provides strategies for handling common errors and failure situations.

### Producing Artifacts for Troubleshooting

There are several types of GemFire files that are critical for troubleshooting.

GemFire logs and statistics are the two most important artifacts used in troubleshooting. In addition, they are required for GemFire system health verification and performance analysis. For these reasons, logging and statistics should always be enabled, especially in production. Save the following files for troubleshooting purposes:

- 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 or stack traces.
- For Linux, you can use gdb to extract a stack from a core file.
- Crash dumps.
- For Windows, save the user mode dump files. Some locations to check for these files:
  - C:\ProgramData\Microsoft\Windows\WER\ReportArchive
  - C:\ProgramData\Microsoft\Windows\WER\ReportQueue
  - C:\Users\UserName\AppData\Local\Microsoft\Windows\WER\ReportArchive
  - C:\Users\UserName\AppData\Local\Microsoft\Windows\WER\ReportQueue

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.



**Note:** 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 config to avoid filling up the disk while including configuration information. Add this line to gemfire.properties:

```
log-level=config
```



**Note:** Running with the log level at fine can impact system performance and fill up your disk.

4. Enable statistics gathering for the distributed system either by modifying gemfire.properties:

```
statistic-sampling-enabled=true
statistic-archive-file=StatisticsArchiveFile.gfs
```

or by using the gfsh alter runtime command:

```
alter runtime --group=myMemberGroup --enable-statistics=true
--statistic-archive-file=StatisticsArchiveFile.gfs
```



**Note:** Collecting statistics at the default sample rate frequency of 1000 milliseconds does not incur performance overhead.

5. Run the application again.

6. 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 these strings:

```
[error
[severe
```

For details on merging log files, see the --merge-log argument for the [export logs](#) on page 691 command.

7. Export and analyze the stack traces on the member or member group where the application is running. Use the gfsh export stack-traces command. For example:

```
gfsh> export stack-traces --file=ApplicationStackTrace.txt --member=member1
```

## Diagnosing System Problems

This section provides possible causes and suggested responses for system problems.

- [Locator does not start](#) on page 607
- [Application or cache server process does not start](#) on page 607
- [Application or cache server does not join the distributed system](#) on page 608
- [Member process seems to hang](#) on page 609
- [Member process does not read settings from the gemfire.properties file](#) on page 609
- [Cache creation fails - must match DOCTYPE root](#) on page 610
- [Cache isn't configured properly](#) on page 610
- [Unexpected results for keySetOnServer and containsKeyOnServer](#) on page 611
- [Data operation returns PartitionOfflineException](#) on page 611
- [Entries are not being evicted or expired as expected](#) on page 611
- [Can't find the log file](#) on page 612
- [OutOfMemoryError](#) on page 612
- [PartitionedRegionDistributionException](#) on page 612
- [PartitionedRegionStorageException](#) on page 613
- [Application crashes without producing an exception](#) on page 613
- [Timeout alert](#) on page 613
- [Member produces SocketTimeoutException](#) on page 614
- [Member logs ForcedDisconnectException, Cache and DistributedSystem forcibly closed](#) on page 614

- [Members cannot see each other on page 614](#)
- [Some new members are not seen by existing members on page 614](#)
- [One part of the distributed system cannot see another part on page 615](#)
- [Data distribution has stopped, though member processes are running on page 615](#)
- [Distributed-ack operations take a very long time to complete on page 615](#)
- [Slow system performance on page 616](#)
- [Can't get Windows performance data on page 616](#)
- [Java applications on 64-bit platforms hang or use 100% CPU on page 616](#)

### **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(TCPGOSS
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.
- If you are using a 64-bit Linux system, check whether your system is experiencing the leap second bug. See [Java applications on 64-bit platforms hang or use 100% CPU on page 616](#) for more information.

### **Application or cache server process does not start**

**Possible Cause 1:** GemFire will not start because it has detected an invalid license property (either `license-data-management` or `license-application-cache` in the `gemfire.properties` file).

Response:

1. Check the GemFire output for a `com.gemstone.gemfire.LicenseException` message. The content of the message will indicate which license property contains an invalid license.

For example, if you have specified an invalid serial number in `license-data-management`, the following message will appear:

The specified serial number "#####-#####-#####-#####-#####" may be expired or invalid for Data Management Node license. Remove serial number from configuration in order to use the default evaluation license.

In this case, you can either:

- remove the serial number from `gemfire.properties` and restart the server. When the server restarts, it will use the default evaluation license; or

- replace the serial number in gemfire.properties with a valid and non-expired serial number. Restart the server and check the logs to make sure the license is valid.
2. If you have specified dynamic in one of the license properties, a message similar to the following may appear if GemFire cannot locate a valid dynamic license:  
Failed to dynamically acquire a Data Management Node license within the 10 second timeout. Consider increasing license-server-timeout or remove "dynamic" from configuration in order to use the default evaluation license.
- In this case, do the following:
- Check to see if there is a serial number file in the serial number directory. If the file exists, verify that the serial number or serial numbers located in the file are valid. See [Local VMware vFabric Directories](#) on page 53 for the appropriate directory on your operating system.
  - Make sure that the GemFire process is running on a vSphere virtual machine that is part of a vSphere installation that includes a vFabric License Server.
  - If you are using a vFabric License Server to manage dynamic licenses, verify that the vFabric License Server is up and running and reachable by the GemFire process.
  - If the vFabric License Server is functioning, try increasing the timeout value in license-server-timeout property of gemfire.properties. Restart the GemFire process.
  - If all else fails, remove the keyword dynamic from license property and reboot the server. When the server restarts, it will use the default evaluation license.

**Possible Cause 2:** 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 JVM heap size. This property is specified on the gfsh command line:  
`gfsh>start server --name=server_name --max-heap=1024m`
- For details, see [JVM Memory Settings and System Performance](#) on page 524.
- 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 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 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 615.
- 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 [Using Bind Addresses](#) on page 169.
- Wrong version of GemFire. 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 . . .
```

### **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 [Disk Storage](#) on page 459. 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 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 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 7.0//EN"  
      "http://www.gemstone.com/dtd/cache7_0.dtd">  
<cache>  
  ...  
</cache>
```

### For clients:

```
<?xml version="1.0"?>  
<!DOCTYPE client-cache PUBLIC  
      "-//GemStone Systems, Inc.//GemFire Declarative Caching 7.0//EN"  
      "http://www.gemstone.com/dtd/cache7_0.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](#) on page 609.
- **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 [Region Data Storage and Distribution](#) on page 205.

## 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:  
[1/10.80.10.64:/export/straw3/users/jpearson/bugfix_Apr10/testCL/hostB/backupDirectory  
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 476.

## 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 [Using Eviction and Expiration Operations](#).

- Partitioned regions—For performance reasons, eviction and expiration behave differently in partitioned regions and can cause entries to be removed before you expect. See [Eviction](#) on page 248 and [Expiration](#) on page 250.

### 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 [Member process does not read settings from the gemfire.properties file](#) on page 609.

### 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 and System Performance](#) on page 524.
- 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 JVMs, 1mb with the 64-bit Sparc 1.4 JVM; and 128k for 1.2 JVMs. 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.

Related Topics
<a href="#">Heap Use and Management</a> on page 453
<a href="#">Memory Requirements for Cached Data</a> on page 989
<a href="#">JVM Memory Settings and System Performance</a> on page 524
<a href="#">General Region Data Management</a> on page 245
Java Memory Management Whitepaper: <a href="http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf">http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf</a>

### 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.

### **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.

### **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 612.

Response: Control memory use by setting entry limits for the regions.

Related Topics
<a href="#">Heap Use and Management</a> on page 453
<a href="#">Memory Requirements for Cached Data</a> on page 989
<a href="#">JVM Memory Settings and System Performance</a> on page 524
<a href="#">General Region Data Management</a> on page 245
Java Memory Management Whitepaper: <a href="http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf">http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf</a>

### **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 above the default 15 seconds.

### **Member produces SocketTimeoutException**

A client, server, gateway sender, or gateway receiver 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 sender and gateway receiver, either in the `cache.xml` file or through the API. For a client/server configuration, adjust the "read-timeout" value as described in [<pool>](#) on page 917 or use the `com.gemstone.gemfire.cache.client.PoolFactory.setReadTimeout` method. For a gateway sender or gateway receiver, see [List item](#). on page 862.

### **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 [Using Bind Addresses](#) on page 169.
- 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 [How Member Discovery Works](#) on page 165.

### **Some new members are not seen by existing members**

If your application creates many, many short-lived members, 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.
- Going forward, set up network monitoring tools to detect these kinds of outages quickly.
- Enable network partition detection.

#### **Related Topics**

[Understanding and Recovering from Network Outages](#) on page 627

### **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 612.

- 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 longer appropriate for the current network traffic.
- 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:
    1. Shut down all the members of the distributed system in the usual order.
    2. Restart the locator process in its new location.
    3. Edit all the `gemfire.properties` files to change this locator's IP address in the `locators` attribute.
    4. 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 [Slow distributed-ack Messages](#) on page 530.

### 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 [Socket Communication](#) on page 530.

### Can't get Windows performance data

Attempting to run performance measurements for GemFire 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.

### Java applications on 64-bit platforms hang or use 100% CPU

If your Java applications suddenly start to use 100% CPU, you may be experiencing the leap second bug. This bug is found in the Linux kernel and can severely affect Java programs. In particular, you may notice that method invocations using `Thread.sleep(n)` where `n` is a small number will actually sleep for much longer period of time than defined by the method. To verify that you are experiencing this bug, check the host's `dmesg` output for the following message:

```
[10703552.860274] Clock: inserting leap second 23:59:60 UTC
```

To fix this problem, issue the following commands on your affected Linux machines:

```
prompt> /etc/init.d/ntp stop
prompt> date -s "$(date)"
```

See the following web site for more information:

<http://blog.wpkg.org/2012/07/01/java-leap-second-bug-30-june-1-july-2012-fix/>

## System Failure and Recovery

This section describes alerts for and appropriate responses to various kinds of system failures. It also helps you plan a strategy for data 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.

## Planning for Data Recovery

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.
- 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 section 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
- Member is Taking Too Long to Respond
- No Locators Can Be Found
- Warning Notifications Before Removal
- Member is Forced Out

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 [Understanding and Recovering from Network Outages](#) on page 627.

### Network Partitioning Detected

Alert:

```
Membership coordinator id has declared that a network partition has occurred.
```

Description:

This alert is issued when network partitioning occurs, followed by this alert on the individual member:

Alert:

```
Exiting due to possible network partition event due to loss of {0} cache
processes: {1}
```

Response:

Check the network connectivity and health of the listed cache processes.

### Member 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



**Note:** 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.

### **Members Failed to Acknowledge Membership View Preparation Message**

Alert:

```
Failed to collect all ACKs (X) for view preparation view id after Y ms,
missing ACKs from list of ids (received=list of ids), local_addr=id
```

Description:

Response:

### **Member Failed to Acknowledge New Membership View**

Alert:

```
Member failed to acknowledge a membership view and network partition
detection is enabled.
```

Description:

Response:

## **Recovering from Application and 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.

### **Recovering from Crashes with a Peer-to-Peer Configuration**

When a member crashes, the remaining members continue operation as though the missing application or cache server had never existed. The recovery process differs according to region type and scope, as well as data redundancy configuration.

The other system members are told that it has left unexpectedly. 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. 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:

- Recovery for Partitioned Regions
- Recovery for Distributed Regions
- Recovery for Regions of Local Scope
- Recovering Data From Disk

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 store.

#### Recovery for Partitioned Regions 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 [Configure High Availability for a Partitioned Region](#) on page 227.

To recover, start a replacement member. The new member regenerates the lost copies and returns them to the configured redundancy level.



**Note:** 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 at the same time 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. If a second goes down at the same time, any data stored by those two members will be lost.

You can also lose access to all copies of your data through network failure. See [Understanding and Recovering from Network Outages](#) on page 627.

#### 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:
[piplin(21944):41927/42712] Data stores successfully allocated:
[piplin(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 stores for the partitioned region.
- Consider starting another member.

## Recovery for Distributed Regions

Restart the process. The system member recreates its cache automatically. If replication is used, data is automatically loaded from the replicated regions, creating an up-to-date cache in sync with the rest of the system.

If you have persisted data but no replicated regions, 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 [Disk Storage](#) on page 459. 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.

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 [Expiration](#) on page 250.

### Disk Recovery for Disk Writing—Synchronous Mode and Asynchronous Mode

#### Synchronous Mode of Disk Writing

Alert 1:

```
DiskAccessException has occurred 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 region may no longer exist on the member. The cache servers and gateway senders may also have been stopped. 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 region may no longer exist on the member. The cache servers and gateway senders may also have been stopped. Recreate the region and restart the cache servers.

### Recovering from Crashes with a Client/Server Configuration

In a client/server configuration, you first make the server available as a member of a distributed system again, and then restart clients as quickly as possible. The client recovers its data from its servers through normal operation.

How well a client/server configuration recovers from application or cache server crashes depends on server availability and on client configuration. Typically, 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 high availability of messaging in the case of client crashes, 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 [Implementing Durable Client/Server Messaging](#) on page 301.

### Recovering from Server Failure

Recovery from server failure has two parts: the server recovers as a member of a distributed system, 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 [Recovering from Crashes with a Peer-to-Peer Configuration](#) on page 621.

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 [Recovering from Crashes with a Peer-to-Peer Configuration](#) on page 621.
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. See [Client/Server Configuration](#) on page 177.

### 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. This is unusual, but might be the case for your system. If the server pools are configured without static server lists, meaning clients use locators to find their servers, 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 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.

### 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.

### Recovery Procedure

To recover from a machine crash:

1. Determine which processes run on this machine.
2. Reboot the machine.
3. If a GemFire locator runs here, start it first.



**Note:** 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 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 stores 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 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. See also [Recovery for Partitioned Regions](#) on page 622.

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 623.

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 627 and [Data Recovery for Distributed Regions](#) on page 627 as appropriate.

The impact of a server crash on its clients depends on whether the installation is configured for highly available servers. For information, see [Recovering from Crashes with a Client/Server Configuration](#) on page 625.

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 626.

## Understanding and Recovering from Network Outages

The safest response to a network outage 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.

- [What Happens During a Network Outage](#) on page 628
- [Recovery Procedure](#) on page 628

- [Effect of Network Failure on Partitioned Regions](#) on page 628
- [Effect of Network Failure on Distributed Regions](#) on page 628
- [Effect of Network Failure on Client/Server Installations](#) on page 628

## What Happens During a Network Outage

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 as explained below.

### Recovery Procedure

To recover from a network outage:

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.

The members recreate the data as they return to active work. For details, see [Recovering from Application and Cache Server Crashes](#) on page 621.

### 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 stores 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.

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 626. 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 625.

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.

# Part 8

---

## Deploying and Running vFabric GemFire

*Deploying and Running vFabric GemFire* describes how to deploy your production system, and manage startup and shutdown.

### Topics:

- Deploying vFabric GemFire Configuration Files
- Managing vFabric GemFire System Output Files
- Starting Up and Shutting Down Your System
- Using the GemFire SHell (gfsh)
- vFabric GemFire Locator Process
- vFabric GemFire Servers
- Using Member Groups
- Deploying Application JARs to vFabric GemFire Members
- Firewall Considerations



# Chapter 48

---

## Deploying vFabric GemFire Configuration Files

You can deploy your vFabric GemFire configuration files in your system directory structure or in jar files. You determine how you want to deploy your configuration files and set them up accordingly.

### Main Steps to Deploying Configuration Files

These are the basic steps for deploying configuration files, with related detail in sections that follow.

1. Determine which configuration files you need for your installation.
2. Place the files in your directories or jar files.
3. For any file with a non-default name or location, provide the file specification in the system properties file and/or in the member CLASSPATH .

### GemFire Configuration Files

You will generally use both of these files, but neither is required.

- **gemfire.properties** . Contains the settings required by members of a distributed system. These settings include licensing, system member discovery, communication parameters, logging, and statistics. See [gemfire.properties and gfsecurity.properties \(GemFire Properties\)](#) on page 851.
- **gfsecurity.properties** . An optional separate file that contains security-related (security-\*) settings that are otherwise defined in `gemfire.properties`. Placing these member properties into a separate file allows you to restrict user access to those specific settings. See [gemfire.properties and gfsecurity.properties \(GemFire Properties\)](#) on page 851.
- **cache.xml** . Declarative cache configuration file. This file contains XML declarations for cache, region, and region entry configuration. You also use it to configure disk stores, database login credentials, server and remote site location information, and socket information. See [cache.xml](#) on page 861.

### Default File Specifications and Search Locations

Each file has a default name, a set of file search locations, and a system property you can use to override the defaults.

To use the default specifications, 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 filenames.



**Note:** If you do not specify an absolute file path and name, the search examines all search locations for the file.

Default File Specification	Search Locations for Relative File Specifications	Available Property for File Specification
gemfire.properties	1. current directory 2. home directory 3. CLASSPATH	As a Java system property, use gemfirePropertyFile
cache.xml	1. current directory 2. CLASSPATH	In gemfire.properties, use the cache-xml-file property

Examples of valid gemfirePropertyFile specifications:

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

For the test1/gfprops specification, if you launch your GemFire system member from /testDir in a Unix file system, GemFire looks for the file in this order until it finds the file or exhausts all locations:

1. /testDir/test1/gfprops
2. <yourHomeDir>/test1/gfprops
3. under every location in your CLASSPATH for test1/gfprops

## Changing the File Specifications

You can change all file specifications in the gemfire.properties file and at the command line.



**Note:** GemFire applications can use the API to pass java.lang.System properties to the distributed system connection. This changes file specifications made at the command line and in the gemfire.properties. You can verify an application's property settings in the configuration information logged at application startup. The configuration is listed when the gemfire.properties log-level is set to config or lower.

This invocation of the application, testApplication.TestApp1, provides non-default specifications for both the cache.xml and gemfire.properties:

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

The gfsh start server command can use the same specifications:

```
gfsh>server 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 inside the gemfire.properties file.



**Note:** Specifications in gemfire.properties files cannot use environment variables.

Example gemfire.properties file with non-default cache.xml specification:

```
#Tue May 09 17:53:54 PDT 2006
mcast-address=224.0.0.250
```

```
mcast-port=10333
locators=
cache-xml-file=/gemfireSamples/examples/dist/cacheRunner/queryPortfolios.xml
```

## Deploying Configuration Files in JAR Files

This section provides a procedure and an example for deploying configuration files in JAR files.

### Procedure

1. Jar the files.
2. Set the vFabric GemFire system properties to point to the files as they reside in the jar file.
3. Include the jar file in your CLASSPATH.
4. Verify the jar file copies are the only ones visible to the application at runtime. GemFire searches the CLASSPATH after searching other locations, so the files cannot be available in the other search areas.
5. Start your application. The configuration file is loaded from the jar file.

### Example of Deploying a Configuration JAR

The following example deploys the cache configuration file, `myCache.xml`, in `my.jar`. The following displays the contents of `my.jar`.

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

In this example, you would perform the following steps to deploy the configuration jar file:

1. Set the system property, `gemfire.cache-xml-file`, to `myConfig/myCache.xml`
2. Set your CLASSPATH to include `my.jar`.
3. Verify there is no file named `myCache.xml` in `./myConfig/myCache.xml`, the current directory location of the file

When you start your application, the configuration file is loaded from the jar file.



# Chapter 49

---

## Managing vFabric GemFire System Output Files

GemFire output files are optional and can become quite large. Work with your system administrator to determine where to place them to avoid interfering with other system activities.

GemFire includes several types of optional output files as described below.

- **Log Files.** Comprehensive logging messages to help you confirm system configuration and to debug problems in configuration and code. Configure log file behavior in the `gemfire.properties` file. See [Logging](#) on page 543.
- **Statistics Archive Files.** Standard statistics for caching and distribution activities, which you can archive on disk. Configure statistics collection and archival in the `gemfire.properties`, `archive-disk-space-limit` and `archive-file-size-limit`. See [gemfire.properties and gfsecurity.properties \(GemFire Properties\)](#) on page 851.
- **Disk Store Files.** Hold persistent and overflow data from the cache. You can configure regions to persist data to disk for backup purposes or overflow to disk to control memory use. The subscription queues that servers use to send events to clients can be overflowed to disk. Gateway sender queues overflow to disk automatically and can be persisted for high availability. Configure these through the `cache.xml`. See [Disk Storage](#) on page 459.



# Chapter 50

---

## Starting Up and Shutting Down Your System

Determine the proper startup and shutdown procedures and write your startup and shutdown scripts.

### Introduction to Startup and Shutdown

Well-designed procedures for starting and stopping your system can speed startup and protect your data.

The processes you need to start and stop include server and locator processes and your other GemFire applications, including clients. The procedures you use depend in part on your system's configuration and the dependencies between your system processes.

Use the following guidelines to create startup and shutdown procedures and scripts. Some of these instructions use the [Using the GemFire SHell \(gfsh\)](#) on page 641.

### Starting Up Your System

You should follow certain order guidelines when starting your GemFire system.

Start server-distributed systems before you start their client applications. In each distributed system, follow these guidelines for member startup:

- Start locators first. See [vFabric GemFire Locator Process](#) on page 737 for examples of locator startup commands.
- Start cache servers before the rest of your processes unless the implementation requires that other processes be started ahead of them. [vFabric GemFire Servers](#) on page 741 for examples of cache server startup commands.
- If your distributed system uses both persistent replicated and non-persistent replicated regions, you should start up all the persistent replicated members in parallel before starting the non-persistent regions. This way, persistent members will not delay their startup for other persistent members with later data.
- If you are running producer processes and consumer or event listener processes, start the producers first. This ensures the consumers and listeners receive all notifications and updates.



**Note:** You can optionally override the default timeout period for shutting down individual processes. This override setting must be specified during member startup. See [Option for System Member Shutdown Behavior](#) on page 639 for details.

### Starting Up After Losing Data on Disk

This information pertains to catastrophic loss of GemFire disk store files. If you lose disk store files, your next startup may hang, waiting for the lost disk stores to come back online. If your system hangs at startup, use the `gfsh command show missing-disk-store` to list missing disk stores and, if needed, revoke missing

disk stores so your system startup can complete. You must use the Disk Store ID to revoke a disk store. These are the two commands:

```
gfsh>show missing-disk-stores

      Disk Store ID          |   Host    |           Directory
-----+-----+
60399215-532b-406f-b81f-9b5bd8d1b55a | excalibur |
/usr/local/gemfire/deploy/disk_store1

gfsh>revoke missing-disk-store --id=60399215-532b-406f-b81f-9b5bd8d1b55a
```



**Note:** This gfsh commands require that you are connected to the distributed system via a JMX Manager node.

## Shutting Down the System

Shut down your GemFire system by using either gfsh's shutdown command or by shutting down individual members one at a time.

### Using the shutdown Command

If you are using persistent regions, (members are persisting data to disk), you should use the gfsh shutdown command to stop the running system in an orderly fashion. This command synchronizes persistent partitioned regions before shutting down, which makes the next startup of the distributed system as efficient as possible.

If possible, all members should be running before you shut them down so synchronization can occur. Shut down the system using the following gfsh command:

```
gfsh>shutdown
```

To shutdown all members after a grace period, specify a time-out option (in seconds).

```
gfsh>shutdown --time-out=60
```



**Note:** The gfsh shutdown command does not shut down GemFire locators. To shut down individual locators in your distributed system, use the stop locator command on each locator.

### Shutting Down System Members Individually

If you are not using persistent regions, you can shut down the distributed system by shutting down each member in the reverse order of their startup. (See [Starting Up Your System](#) on page 637 for the recommended order of member startup.)

Shut down the distributed system members according to the type of member. For example, use the following mechanisms to shut down members:

- Use the appropriate mechanism to shut down any GemFire-connected client applications that are running in the distributed system.
- Shut down any cache servers. To shut down a cache server, issue the following gfsh command:

```
gfsh>stop server --name=<...>
```

See [stop server](#) on page 731 for more details.

- Shut down any locators. To shut down a locator, issue the following gfsh command:

```
gfsh>stop locator --name=<...>
```

See [stop locator](#) on page 730 for more details.

### Option for System Member Shutdown Behavior

The DISCONNECT\_WAIT command line argument sets the maximum time for each individual step in the shutdown process. If any step takes longer than the specified amount, it is forced to end. Each operation is given this grace period, so the total length of time the cache member takes to shut down depends on the number of operations and the DISCONNECT\_WAIT setting. During the shutdown process, GemFire produces messages such as:

```
Disconnect listener still running
```

The DISCONNECT\_WAIT default is 10000 milliseconds.

To change it, set this system property on the Java command line used for member startup. For example:

```
gfsh>start server --J=-DDistributionManager.DISCONNECT_WAIT=<milliseconds>
```

Each process can have different DISCONNECT\_WAIT settings.



# Chapter 51

---

## Using the GemFire Shell (gfsh)

vFabric GemFire gfsh (pronounced "gee-fish") provides a single, intuitive command-line interface from which you can launch, manage, and monitor vFabric GemFire processes, data, and applications.

### What You Can Do with gfsh

gfsh supports the administration, debugging, and deployment of GemFire processes and applications.

With gfsh, you can:

- Start and stop GemFire processes, such as locators and cache servers
- Start and stop gateway sender and gateway receiver processes
- Deploy applications
- Create and destroy regions
- Execute functions
- Manage disk stores
- Import and export data
- Monitor GemFire processes
- Launch GemFire monitoring tools

The gfsh command line interface lets developers spend less time configuring cache instance XML, properties, logs, and statistics. gfsh commands generate reports; capture cluster-wide statistics; and support the export of statistics, logs, and configurations. Like Spring Roo, gfsh features command completion (so you do not have to know the syntax), context-sensitive help, scripting, and the ability to invoke any commands from within an application by using a simple API. The gfsh interface uses JMX/RMI to communicate with GemFire processes.

The GemFire 7.0 version of gfsh supports all of the commands that existed in the previous version of gfsh (released with GemFire 6.6) and now includes functionality that was previously found in the `gemfire` and `cacheserver` scripts.

### Starting gfsh

Before you start gfsh, confirm that you have set the GEMFIRE and either the GF\_JAVA or JAVA\_HOME environment variables to the proper locations for your deployment, or that your PATH variable includes the java executable.

See [Installing vFabric GemFire](#) on page 27 for more information.

To manage servers and locators properly, gfsh requires that tools.jar (provided with the JDK) exist in the CLASSPATH.



**Note:** On Windows, you must have the JAVA\_HOME variable set properly to use start, stop and status commands for both locators and servers.

To launch the gfsh command-line interface, execute the following command at the prompt on any machine that is currently installed with GemFire:

## **Start gfsh on Windows:**

`<product_directory>\bin\gfsh.bat`

where `<product_directory>` corresponds to the location where you installed GemFire.

## Start gfsh on Unix:

`<product_directory>/bin/gfsh`

where `<product_directory>` corresponds to the location where you installed GemFire. If you installed GemFire using RPM, the default product directory installation location is `/opt/vmware/vfabric-gemfire`. Upon execution, the `gfsh` script appends the required GemFire and JDK .jar libraries to your existing CLASSPATH.

If you have successfully started gfsh, the gfsh splash screen and prompt appears.

c:\vFabricGemFire70\Latest>gfsh.bat

## Monitor and Manage GemFire

You can also run some gfsh commands directly within your terminal without entering a gfsh prompt. For example, on Unix/Linux you could enter:

```
$ gfsh start server --name=server1
```

or on Windows:

```
prompt> gfsh start server --name=server1
```

See [Creating and Running gfsh Command Scripts](#) on page 733 for more information.

# Basic Shell Features and Command-Line Usage

The `gfish` utility provides useful features for a shell environment, including command auto-complete, preserved command history, and delimiting of multi-line commands. Context-sensitive help is available by command and by topic.

This section describes these features.

To view a list of available gfh commands, press Tab at an empty prompt.

The list of commands you see depends on whether you are connected to a GemFire distributed system. If you are not connected, you see a list of local commands that are available.

qfsh>

```
alter disk-store          compact offline-disk-store
connect                  debug
describe connection       describe offline-disk-store
disconnect               echo
encrypt password         exit
help                     hint
history                 run
set variable              sleep
start data-browser        start jconsole
start ivisualvm           start locator
```

start pulse	start server
start vsd	status locator
status server	stop locator
stop server	upgrade offline-disk-store
validate offline-disk-store	version

### Use the **hint** command to get information on a particular topic.

The hint command displays a one-line description and associated commands for a specified topic. For example, typing **hint data** returns a description of the "data" topic and all the possible actions available for data:

```
gfsh>hint data
User data as stored in regions of the GemFire distributed system.

create index   : Create an index that can be used when executing queries.
destroy index  : Destroy/Remove the specified index.
export data    : Export user data from a region to a file.
get           : Display an entry in a region. If using a region whose key
and value classes have been set, then specifying --key-class and
--value-class is unnecessary.
import data   : Import user data from a file to a region.
list indexes  : Display the list of indexes created for all members.
locate entry   : Identifies the location, including host, member and region,
of entries that have the specified key.
put           : Add/Update an entry in a region. If using a region whose
key and value classes have been set, then specifying --key-class and
--value-class is unnecessary.
query         : Run the specified OQL query as a single quoted string and
display the results in one or more pages. Limit will default to the value
stored in the "APP_FETCH_SIZE" variable. Page size will default to the value
stored in the "APP_COLLECTION_LIMIT" variable.
rebalance     : Rebalance partitioned regions. The default is for all
partitioned regions to be rebalanced.
remove        : Remove an entry from a region. If using a region whose key
class has been set, then specifying --key-class is unnecessary.
```

Type **hint server** to view a description of servers in GemFire and a list of commands available for managing and monitoring them.

```
gfsh>hint server
A server is GemFire cluster member which holds a GemFire cache. Depending
on the topology used it can refer to either a system that responds to client
requests or a system that is only a peer to other members.

describe member : Display information about a member, including name, id,
groups, regions, etc.
export logs    : Export the log files for a member or members.
list members   : Display all or a subset of members.
start server   : Start a GemFire Cache Server.
status server  : Display the status of a GemFire Cache Server.
stop server    : Stop a GemFire Cache Server.
```

### To view a list of hint topics, type **hint**.

```
gfsh>hint
Hints are available for following topics. Use "hint <topic-name>" for a
specific hint.
Configuration
Data
Debug-Utility
Disk Store
Function Execution
GFSH
```

```
GemFire
Help
JMX
Lifecycle
Locator
Management-Monitoring
Manager
Region
Server
Statistics
WAN
```

### Use the help command to get information on a particular command.

Depending on the command, typing `help <command-name>` displays usage information for that particular command or a list of commands related to the command.

For example, type `help start` to display a list of start commands with short descriptions. The descriptions indicate whether the command is available, depending on your connection status. In the following example, you are not currently connected to a vFabric GemFire distributed system (via connection to a JMX Manager node), so some start commands are unavailable.

```
gfsh>help start
start data-browser (Available)
    Start Data Browser in a separate process.
start gateway-receiver (Not Available)
    Start the Gateway Receiver on a member or members.
start gateway-sender (Not Available)
    Start the Gateway Sender on a member or members.
start jconsole (Available)
    Start the JDK's JConsole tool in a separate process. JConsole will be
    launched, but connecting to GemFire must be done manually.
start jvisualvm (Available)
    Start the JDK's Java VisualVM (jvisualvm) tool in a separate process.
Java
    VisualVM will be launched, but connecting to GemFire must be done
manually.
start locator (Available)
    Start a Locator.
start pulse (Available)
    Open a new window in the default Web browser with the URL for the Pulse
    application.
start server (Available)
    Start a GemFire Cache Server.
start vsd (Available)
    Start VSD in a separate process.
```

In the following example, typing `help export data` displays the command name, availability (depending on your connection status), a description, its syntax, and a list of all its required and optional parameters.

```
gfsh>help export data
NAME
    export data
IS AVAILABLE
    false
SYNOPSIS
    Export user data from a region to a file.
SYNTAX
    export data --region=value --file=value --member=value
PARAMETERS
```

```

region
    Region from which data will be exported.
    Required:true
file
    File to which the exported data will be written.
The file must have an extension of ".gfd".
    Required:true
member
    Name/Id of a member which hosts the region.
The data will be exported to the specified file on the host
where the member is running.
    Required:true

```

### Use the Tab key to auto-complete a command or trigger possible completions.

Use the Tab key after entering a partial command to trigger auto-completion of the command or a list of possible command completions. For example, hitting Tab after typing hint displays all available topics:

```

gfsh>hint

Configuration          Data           Debug-Utility
Disk Store            Function Execution GFSH
GemFire               Help            JMX
Lifecycle             Locator          Management-Monitoring
Manager               Region           Server
Statistics            WAN

```

Hitting tab after typing hint d displays the topics available that begin with 'd':

```

gfsh>hint d

data          debug-Utility   disk Store

```

Auto-completion also supplies available parameters and arguments to commands; for example, hitting tab after typing start will list all the processes that can be started.

```

gfsh>start

start data-browser  start jconsole      start jvisualvm
start locator       start pulse        start server
start vsd

```

Hitting tab after start locator will populate the required --name parameter after the command.



**Note:** The available commands that are listed varies depending on whether you are currently connected to a distributed system.

### Access command history with the Up arrow.

Access a command in your shell history by scrolling through previous commands with the Up arrow.

### Delimit multi-line commands with \.

When entering long commands, you can break the command line up by using the \ character as a delimiter. For example:

```

gfsh>create region --name=region1 \
--type=REPLICATE_PERSISTENT \
--cache-writer=com.gemstone.gemfire.examples.MyCacheWriter \
--group=Group1 --disk-store=DiskStore1

```

## Tutorial: Performing Common Tasks with gfsh

This topic takes you through a typical sequence of tasks that you execute after starting gfsh.

If you haven't already, start a gfsh prompt. See [Starting gfsh](#) on page 641 for details.

**Step 1: Start up a locator.** Enter the following command:

```
gfsh>start locator --name=locator1
```

The following output appears:

```
gfsh>start locator --name=locator1
Starting a Locator in /home/stymon/locator1 on pickle[10334] as locator1...
.....
Locator in /home/stymon/locator1 on pickle[10334] as locator1 is currently
online.
Process ID: 3464
Uptime: 4 seconds
GemFire Version: 7.0.1
Java Version: 1.6.0_38
Log File: /home/stymon/locator1/locator1.log
JVM Arguments: -Dgemfire.launcher.registerSignalHandlers=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path:
[REDACTED]

Successfully connected to: [host=pickle, port=1099]
```

In your file system, examine the folder location where you executed gfsh. Notice that the `start locator` command has automatically created a working directory (using the name of the locator), and within that working directory, it has created a log file, a status file, and a .pid (containing the locator's process ID) for this locator.

In addition, because no other JMX Manager exists yet, notice that gfsh has automatically started an embedded JMX Manager on port 1099 within the locator and has connected you to that JMX Manager.

**Step 2: Examine the existing gfsh connection.**

In the current shell, type the following command:

```
gfsh>describe connection
```

If you are connected to the JMX Manager started within the locator that you started in Step 1, the following output appears:

```
Connection Endpoints
-----
pickle[1099]
```

Notice that the JMX Manager is on 1099 whereas the locator was assigned the default port of 10334.

**Step 3: Connect to the same locator/JMX Manager from a different terminal.**

This step shows you how to connect to a locator/JMX Manager. Open a second terminal window, and start a second gfsh prompt. Type the same command as you did in Step 2 in the second prompt:

```
gfsh>describe connection
```

This time, notice that you are not connected to a JMX Manager, and the following output appears:

```
gfsh>describe connection
Connection Endpoints
-----
Not connected
```

Type the following command in the second gfsh terminal:

```
gfsh>connect
```

The command will connect you to the currently running local locator that you started in Step 1.

```
gfsh>connect
Connecting to Locator at [host=pickle, port=10334] ..
Connecting to Manager at [host=pickle, port=1099] ..
Successfully connected to: [host=pickle, port=1099]
```

Note that if you had used a custom --port when starting your locator, or you were connecting from the gfsh prompt on another member, you would also need to specify --locator=host[port] when connecting to the distributed system. For example (type disconnect first if you want to try this next command):

```
gfsh>connect --locator=pickle[10334]
Connecting to Locator at [host=pickle, port=10334] ..
Connecting to Manager at [host=pickle, port=1099] ..
Successfully connected to: [host=pickle, port=1099]
```

Another way to connect your gfsh prompt to the distributed system would be to connect to directly to the JMX Manager running inside the locator. For example (type disconnect first if you want to try this next command):

```
gfsh>connect --jmx-manager=pickle[1099]
Connecting to Manager at [host=pickle, port=1099] ..
Successfully connected to: [host=pickle, port=1099]
```

**Step 4: Disconnect and close the second terminal window.** Type the following commands to disconnect and exit the second gfsh prompt:

```
gfsh>disconnect
Disconnecting from: GemFireStymon[1099]
Disconnected from : GemFireStymon[1099]
gfsh>exit
```

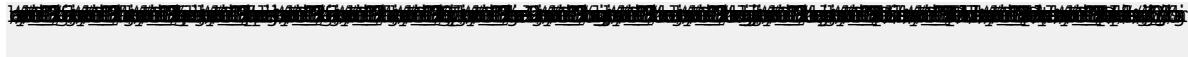
Close the second terminal window.

**Step 5: Start up a server.** Return to your first terminal window, and start up a cache server that uses the locator you started in Step 1. Type the following command:

```
gfsh>start server --name=server1 --locator=pickle[10334]
```

If the server starts successfully, the following output appears:

```
gfsh>start server --name=server1 --locator=pickle[10334]
Starting a Cache server in /home/stymon/server1 on pickle[40404] as
server1...
.....
Server in /home/stymon/server1 on pickle[40404] as server1 is currently
online.
Process ID: 3813
Uptime: 3 seconds
GemFire Version: 7.0.1
Java Version: 1.6.0_38
Log File: /home/stymon/server1/server1.log
JVM Arguments: -Dgemfire.default.locators=192.168.129.144[10334]
-XX:OnOutOfMemoryError="kill -9 %p"
-Dgemfire.launcher.registerSignalHandlers=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path:
```



In your file system, examine the folder location where you executed gfsh. Notice that just like the start locator command, the start server command has automatically created a working directory (named after the server), and within that working directory, it has created a log file and a .pid (containing the server's process ID) for this cache server. In addition, it has also written files that are used for licensing.

**Step 6: List members.** Use the list members command to view the current members of the GemFire distributed system you have just created.

```
gfsh>list members
Name      | Id
----- | -----
server1  | pickle(server1:3813)<v16>:3789
locator1 | pickle(locator1:3464:locator)<v13>:22473
```

**Step 7: View member details by executing the describe member command.**

```
gfsh>describe member --name=server1
Name      : server1
Id        : pickle(server1:3813)<v16>:3789
Host      : pickle
Regions   :
PID       : 3813
Groups    :
Used Heap : 9M
Max Heap  : 485M
Working Dir: /home/stymon/server1
Log file   : /home/stymon/server1/server1.log
Locators   : 192.168.129.144[10334]

Cache Server Information
Server Bind      : null
Server Port      : 40404
Running          : true
Client Connections: 0
```

Note that no regions have been assigned to this member yet.

**Step 8: Create your first region.** Type the following command followed by the tab key:

```
gfsh>create region --name=region1 --type=
```

A list of possible region types appears, followed by the partial command you entered:

```
gfsh>create region --name=region1 --type=
PARTITION
PARTITION_REDUNDANT
PARTITION_PERSISTENT
PARTITION_REDUNDANT_PERSISTENT
PARTITION_OVERFLOW
PARTITION_REDUNDANT_OVERFLOW
PARTITION_PERSISTENT_OVERFLOW
PARTITION_REDUNDANT_PERSISTENT_OVERFLOW
PARTITION_HEAP_LRU
PARTITION_REDUNDANT_HEAP_LRU
REPLICATE
REPLICATE_PERSISTENT
REPLICATE_OVERFLOW
REPLICATE_PERSISTENT_OVERFLOW
REPLICATE_HEAP_LRU
LOCAL
```

```

LOCAL_PERSISTENT
LOCAL_HEAP_LRU
LOCAL_OVERFLOW
LOCAL_PERSISTENT_OVERFLOW
PARTITION_PROXY
PARTITION_PROXY_REDUNDANT
REPLICATE_PROXY

gfsh>create region --name=region1 --type=

```

Complete the command with the type of region you want to create. For example, create a local region:

```

gfsh>create region --name=region1 --type=LOCAL
Member | Status
----- | -----
server1 | Region "/region1" created on "server1"

```

Because only one server is in the distributed system at the moment, the command creates the local region on server1.

**Step 9: Start another server.** This time specify a --server-port argument with a different server port because you are starting a cache server process on the same host machine.

```

gfsh>start server --name=server2 --server-port=40405
Starting a Cache server in /home/stymon/server2 on pickle[40405] as
server2...
.....
Server in /home/stymon/server2 on pickle[40405] as server2 is currently
online.
Process ID: 3967
Uptime: 3 seconds
GemFire Version: 7.0.1
Java Version: 1.6.0_38
Log File: /home/stymon/server2/server2.log
JVM Arguments: -Dgemfire.default.locators=192.168.129.144[10334]
-XX:OnOutOfMemoryError="kill -9 %p"
-Dgemfire.launcher.registerSignalHandlers=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path:

```

#### Step 10: Create a replicated region.

```

gfsh>create region --name=region2 --type=REPLICATE
Member | Status
----- | -----
server1 | Region "/region2" created on "server1"
server2 | Region "/region2" created on "server2"

```

#### Step 11: Create a partitioned region.

```

gfsh>create region --name=region3 --type=PARTITION
Member | Status
----- | -----
server1 | Region "/region3" created on "server1"
server2 | Region "/region3" created on "server2"

```

#### Step 12: Create a replicated, persistent region.

```

gfsh>create region --name=region4 --type=REPLICATE_PERSISTENT
Member | Status
----- | -----

```

```
server1 | Region "/region4" created on "server1"
server2 | Region "/region4" created on "server2"
```

**Step 13: List regions.** A list of all the regions you just created displays.

```
gfsh>list regions
List of regions
-----
region1
region2
region3
region4
```

**Step 14: View member details again by executing the `describe member` command.**

```
gfsh>describe member --name=server1
Name      : server1
Id        : pickle(server1:3813)<v16>:3789
Host      : pickle
Regions   : region4
           region3
           region2
           region1
PID       : 3813
Groups    :
Used Heap : 12M
Max Heap  : 485M
Working Dir: /home/stymon/server1
Log file   : /home/stymon/server1/server1.log
Locators   : 192.168.129.144[10334]

Cache Server Information
Server Bind      : null
Server Port      : 40404
Running          : true
Client Connections: 0
```

Notice that all the regions that you created now appear in the "Regions" section of the member description.

```
gfsh>describe member --name=server2
Name      : server2
Id        : pickle(server2:3967)<v17>:19343
Host      : pickle
Regions   : region4
           region3
           region2
PID       : 3967
Groups    :
Used Heap : 4M
Max Heap  : 485M
Working Dir: /home/stymon/server2
Log file   : /home/stymon/server2/server2.log
Locators   : 192.168.129.144[10334]

Cache Server Information
Server Bind      : null
Server Port      : 40405
Running          : true
Client Connections: 0
```

Notice that the second server only has region2, region3, and region4 on it.

**Step 14: Put data in a local region.** Enter the following put command:

```
gfsh>put --key='123' --value='ABC' --region=region1
Result      : true
Key Class   : java.lang.String
Key         : ('123')
Value Class : java.lang.String
Old Value   : <NULL>
```

**Step 15: Put data in a replicated region.** Enter the following put command:

```
gfsh>put --key='123abc' --value='Hello World!!' --region=region2
Result      : true
Key Class   : java.lang.String
Key         : ('123abc')
Value Class : java.lang.String
Old Value   : <NULL>
```

**Step 16: Retrieve data.** You can use `locate entry`, `query` or `get` to return the data you just put into the region.

For example, using the `get` command:

```
gfsh>get --key='123' --region=region1
Result      : true
Key Class   : java.lang.String
Key         : ('123')
Value Class : java.lang.String
Value       : ABC
```

For example, using the `locate entry` command:

```
gfsh>locate entry --key='123abc' --region=region2
Result      : true
Key Class   : java.lang.String
Key         : ('123abc')
Locations Found : 2

MemberName | MemberId
----- | -----
server2    | pickle(server2:3967)<v17>:19343
server1    | pickle(server1:3813)<v16>:3789
```

Notice that because the entry was put into a replicated region the entry is located on both distributed system members.

For example, using the `query` command:

```
gfsh>query --query='SELECT * FROM /region2'

Result      : true
startCount  : 0
endCount    : 20
Rows        : 1

Result
-----
('Hello World!!')

NEXT_STEP_NAME : END
```

**Step 17: Export your configurations and data.** When you exit the `gfsh` shell, you lose any locator and server configurations and any data that you have created within the shell. To save your configuration and data, you can

use the `export config` and `export data` commands to save your settings as files that you can later copy the configuration files into the working directories of other members. For example:

```
gfsh>export config
Downloading Cache XML file: /home/stymon/./server1-cache.xml
Downloading properties file: /home/stymon/./server1-gf.properties
Downloading Cache XML file: /home/stymon/./server2-cache.xml
Downloading properties file: /home/stymon/./server2-gf.properties
```

Then in a regular terminal, you can simply use your operating system's copy command to put copies of those configuration files into the working directories of the members. For example:

```
prompt# cp server1-gf.properties server1/gemfire.properties
prompt# cp server1-cache.xml server1/cache.xml
prompt# cp server2-gf.properties server2/gemfire.properties
prompt# cp server2-cache.xml server2/cache.xml
```

The next time you start the servers, you only need to specify their name. The `--server-port` specification and regions that you defined in these previous steps will already be created for you.

To export server region data, use the `export data` command. For example:

```
gfsh>export data --region=region1 --file=region1.gfd --member=server1
```

You can later use the `import data` command to import that data into the same region on another member.

## Configuring the gfsh Environment

The `gfsh.bat` and `gfsh` bash script automatically append the required GemFire and JDK .jar libraries to your existing CLASSPATH. There are user-configurable properties you can set for security, environment variables, logging, and troubleshooting.

- [JAR Libraries in CLASSPATH](#) on page 652
- [Configuring gfsh Security](#) on page 653
- [Configuring gfsh Environment Variables](#) on page 653
- [gfsh Logging and Troubleshooting](#) on page 653
- [Command History and gfsh.history](#) on page 654
- [Machine Hostname](#) on page 653

### JAR Libraries in CLASSPATH

The .jar libraries that need to be in your CLASSPATH to run `gfsh` commands include:

- `gemfire.jar`
- `antlr.jar`
- `gfsh-dependencies.jar`
- `tomcat-embed-core.jar`
- `tomcat-embed-logging-juli.jar`
- `tomcat-embed-jasper.jar`
- `ecj-3.7.2.jar`
- `spring-asn-3.1.1.RELEASE.jar`

These JAR files are packaged in your GemFire installation in the `$GEMFIRE/lib` (or on Windows, the `%GEMFIRE%\lib`) directory.

The `gfsh` command-line utility also requires that the `tools.jar` included with JDK is added to your CLASSPATH. The `gfsh` scripts add `$JAVA_HOME/lib/tools.jar` or `%JAVA_HOME%\lib\tools.jar` to your CLASSPATH.



**Note:** If you wish to start a [GemFire Pulse](#) on page 819 application server from the gfsh interface, you must also add \$GEMFIRE/lib/commons-logging.jar to your CLASSPATH. The tomcat-\* JAR files listed above (also used by GemFire Pulse) are automatically added for you by the gfsh scripts, however commons-logging.jar must be added manually.

## Machine Hostname

On some operating systems, you may need to ensure that the hostname of your machine is configured in your system hosts file. For example, on MacOS you may need to map your machine's hostname to your IP address in the /etc/hosts file in order for gfsh and Pulse to operate correctly.

## Configuring gfsh Security

Since gfsh must connect to a JMX Manager member to run certain commands (namely those commands that manage and monitor other members), JMX Manager configuration properties can affect gfsh security. In gemfire.properties, the following GemFire properties can affect gfsh connection settings to the JMX Manager:

- jmx-manager-ssl
- jmx-manager-port
- jmx-manager-password-file
- jmx-manager-access-file

You may also need to verify that the ports are available and open to client connections. See [Configuring a JMX Manager](#) on page 435 for details on these security properties.

## Configuring gfsh Environment Variables

In addition, you can set several gfsh-specific preset SHELL variables by using the `set variable` command. For example, you can set gfsh to run in quiet mode. Not all gfsh variables are modifiable; user-configurable variables include:

- APP\_FETCH\_SIZE
- APP\_COLLECTION\_LIMIT
- APP QUIET\_EXECUTION

See [Useful gfsh Shell Variables](#) on page 654 for more information.

## gfsh Logging and Troubleshooting

gfsh writes several log files while running. Useful log files include:

- **gfsh-YYYY-MM-DD\_HH-MM-SS.log.** Records the events of an individual gfsh session. It includes environment information, such as Java and system information, and detailed command execution. The log file is written to the current working directory where you have executed the gfsh or gfsh.bat script. The desired log level for this file can be set using `gfsh.log-level` system property. Add it to the `JAVA_ARGS` environment before starting gfsh. You cannot modify the output filename or the location where the log file is written.
- <locator-name>.log. Details a locator's configuration (including all gemfire.properties) and all activity that occurs on the locator after startup. This log file is written to a directory that is named after the locator. For example, if you start a locator named locator1, the file is written as locator1.log in the <product\_dir>/locator1 directory.
- **vf.gf.locator.pid.** Contains the process ID of the locator. You can use the PID to stop or view the status of this locator. This file is written to the same directory location as the locator's log file .

- <server-name>.log. Details a server's configuration (including all gemfire.properties) and all activity that occurs on the server after startup. This log file is written to a directory that is named after the server. For example, if you start a server named server1, the file is written as server1.log in the <product\_dir>/server1 directory. If you stop and start the server with an identical name, the older log files are kept in the same directory but renamed for versioning purposes.
- vf.gf.server.pid. Contains the process ID of the server. You can use the PID to stop or view the status of this server. This file is written to the same location as the server log file.

In addition, gfsh records a history of commands in the gfsh.history file, which you can use to create scripts or review past commands.

### Command History and gfsh.history

A history of commands that have been executed successfully is logged in .gfsh.history file in the home directory of the user running gfsh. You can also export a history file by using the history --file=your\_file\_name command.

### JMX Manager Update Rate and System Monitoring

When you perform data operations (such as put) and then monitor the state of the system (such as using the gfsh show metrics command or GemFire Pulse), the monitored system may not immediately reflect the most recent operations. For example, if you perform a put operation and then immediately execute the show metrics gfsh command, you may not see the correct number of entries in the region. The management layer updates every 2 seconds. Wait a few seconds after performing operational activity to see the most accurate results.

You can modify the jmx-manager-update-rate property in gemfire.properties to increase or decrease the rate (specified in milliseconds) at which updates are pushed to the JMX Manager. This property setting should be greater than or equal to the statistic-sample-rate. You may want to increase this rate if you are experiencing performance issues; however, setting this value too high will cause stale values to be seen in gfsh and GemFire Pulse.

### Useful gfsh Shell Variables

You can use the built-in gfsh shell variables in scripts.

You can also use the set variable command to modify shell behavior or to define your own variables.

To see a list of all gfsh shell variables and their current values, use the following command:

```
gfsh>echo --string=$*
```

To obtain the current value of an existing variable, use the following command syntax (the variable must be enclosed in braces):

```
gfsh>echo --string=${VARIABLE}
```

For example:

```
gfsh>echo --string=${SYS_CLASSPATH}
```

### System Variables

SYS_CLASSPATH	CLASSPATH of the gfsh JVM (read only).
SYS_GEMFIRE_DIR	Product directory where GemFire has been installed (read only).
SYS_HOST_NAME	Host from which gfsh is started (read only).
SYS_JAVA_VERSION	Java version used (read only).
SYS_OS	OS name (read only).

SYS_OS_LINE_SEPERATOR	Line separator (\ or ^) variable that you can use when writing gfsh scripts. (read only).
SYS_USER	User name (read only).
SYS_USER_HOME	User's home directory (read only).

## GFSH Environment Variables

APP_COLLECTION_LIMIT	Number of items in the embedded collection of a result to be iterated Values: 1-100. Default value is 20.
APP_FETCH_SIZE	Fetch size to be used while querying. Values: 0 - 2147483647. Default value is 1000.
APP_LAST_EXIT_STATUS	Last command exit status. Similar to \$? (Unix) and %errorlevel% (Windows). Values: 0 (successful), 1 (error), 2(crash) (read only).
APP_LOG_FILE	Path and name of current gfsh log file (read only).
APP_NAME	Name of the application-- "gfsh" (read only).
APP_PWD	Current working directory where gfsh was launched (read only).
APP_QUERY_RESULTS_DISPLAY_MODE	Toggle the display mode for returning query results. Values: table or catalog. Default value is table.   <b>Note:</b> This variable setting is not applied in this release.
APP QUIET_EXECUTION	Whether the execution should be in quiet mode. Values (case insensitive): true, false. Default value is false.

## Quick Reference of gfsh Commands by Functional Area

This quick reference sorts all commands into functional areas.

### Basic GemFire SHell Commands

#### debug

- availability: online or offline
- arguments:

```
debug --state=on_or_off
```

- examples:

```
debug --state=on
debug --state=OFF
```

#### echo

- availability: online or offline
- arguments:

```
echo [--string=value_to_be_echoed]
```

- examples:

```
echo --string="Hello, this is ${SYS_USER}"
echo --string=${APP_FETCH_SIZE}
```

#### encrypt password

- availability: online or offline

- arguments:

```
encrypt password --password=password_to_encrypt
```

- examples:

```
encrypt password --password=my_password
```

### **exit (or quit)**

- availability: online or offline
- arguments: none
- examples:

```
exit  
quit
```

### **help**

- availability: online or offline
- arguments:

```
help [command]
```

- examples:

```
help connect  
help start  
help start server
```

### **hint**

- availability: online or offline
- arguments:

```
hint [topic]
```

- examples:

```
hint  
hint Data  
hint Function Execution
```

### **history**

- availability: online or offline
- arguments:

```
history [--file=history_filename_to_write]
```

- examples:

```
history  
history --file=my_gfsh_history.txt  
history --file=./mycommands.gfsh
```

### **run**

- availability: online or offline

 **Note:** Commands specified in the file may have their own online or offline connection requirements.

- arguments:

```
run --file=value [--quiet(=true_or_false)?]
[--continue-on-error(=true_or_false)?]
```

- examples:

```
run --file=gfsh_script.txt
run --file=gfsh_script.txt --quiet --continue-on-error
```

### **set variable**

- availability: online or offline

- arguments:

```
set variable --name=variable_name --value=variable_value
```

- examples:

```
set variable --name=APP_FETCH_SIZE --value=1200
set variable --name=MY_VARIABLE --value=GemFire
```

### **sleep**

- availability: online or offline

- arguments:

```
sleep [--time=value in seconds]
```

- examples:

```
sleep
sleep --time=60
```

### **version**

- availability: online or offline

- arguments:

```
version [--full]
```

- examples:

```
version
version --full
```

## **Configuration Commands**

### **alter runtime**

- availability: online

- arguments:

```
alter runtime [--member=name_or_id_of_member] [--group=member_group_name]
[--archive-disk-space-limit=limit_in_megabytes]
[--archive-file-size-limit=limit_in_megabytes]
[--log-disk-space-limit=limit_in_megabytes]
[--log-file-size-limit=limit_in_megabytes]
[--log-level=none_error_info_config_fine_finer_or_finest]
[--statistic-archive-file=filename_where_statistics_are_written]
[--statistic-sample-rate=rate_in_milliseconds]
[--enable-statistics=true_or_false]
```

- examples:

```
alter runtime --member=member1 --enable-statistics=false
```

### **describe config**

- availability: online
- arguments:

```
describe config --member=membername_or_id [--hide-defaults(=true_or_false)?]
```

- examples:

```
describe config --member=member1  
describe config --member=member1 --hide-defaults
```

### **export config**

- availability: online
- arguments:

```
export config [--member=membername_or_id(,additional_membername_or_id)*]  
[--group=groupname(,additional_groupname)*] [--dir=target_dir_name]
```

- examples:

```
export config --member=member1
```

## **Data Commands**

### **export data**

- availability: online
- arguments:

```
export data --region=region_to_export --file=target_file_name.gfd  
--member=membername_or_id_hosting_region
```

- examples:

```
export data --region=MyRegion --file=MyRegionExport.gfd --member=server1
```

### **get**

- availability: online
- arguments:

```
get --key=value --region=region_name [--key-class=object_type]  
[--value-class=object_type]
```

- examples:

```
get --key='id'
```

### **import data**

- availability: online
- arguments:

```
import data --region=region_name --file=file_to_import.gfd  
--member=membername
```

- examples:

```
import data --region=DevRegion --file=MyRegionExport.gfd --member=server2
```

### **locate entry**

- availability: online

- arguments:

```
locate entry --key=value --region=value [--key-class=value]
[--value-class=value] [--recursive=value]
```

- examples:

```
locate entry --key='id'
```

### **put**

- availability: online

- arguments:

```
put --key=value --value=value [--region=value]
[--key-class=value] [--value-class=value] [--skip-if-exists=value]
```

- examples:

```
put --key='id' --value='123'
```

### **query**

 **Note:** This command should not be executed from gfsh if the objects being queried contain cyclic references.

- availability: online

- arguments:

```
query --query=select_statement [--step-name=step_name]
[--interactive=true_or_false]
```

- examples:

```
query --query="SELECT * FROM /exampleRegion WHERE id < 10"
```

## **Deployment Commands**

### **deploy**

- availability: online

- arguments:

```
deploy [--group=target_group_name(,target_group_name)*] [--jar=jar_filename]
[--dir=jar_directory]
```

- examples:

```
deploy --jar=MyApplication.jar
deploy --dir=ApplicationDir
```

### **list deployed**

- availability: online

- arguments:

```
list deployed [--group=member_groupname(,member_groupname)*]
```

- examples:

```
list deployed  
list deployed --group=MemberGroup1
```

### **undeploy**

- availability: online

- arguments:

```
undeploy [--group=target_member_groupname(,target_member_groupname)*]  
[--jar=jar_filename(,jar_filename)*]
```

- examples:

```
undeploy  
undeploy --jar=MyApplication.jar
```

## **Disk Store Commands**

### **alter disk-store**

- availability: online

- arguments:

```
alter disk-store --name=diskstore_name --region=region_name  
--disk-dirs=diskstore_directory(,diskstore_directory)*  
[--lru-algorithm=none_or_lru-entry-count_or_lru-heap-percentage_or_lru-memory-size]  
[--lru-action=value]  
[--lru-limit=none_or_overflow-to-disk_or_local-destroy]  
[--concurrency-level=maximum_number_of_threads]  
[--initial-capacity=number_of_entries]  
[--load-factor=floating_point_between_0_and_1]  
[--enable-statistics=true_or_false]  
[--remove(=true_or_false)?]
```

- examples:

```
alter disk-store --name=Disk1 --region=MyRegion --disk-dirs=/disks/disk1
```

### **backup disk-store**

- availability: online

- arguments:

```
backup disk-store --dir=disk_store_directory  
[--baseline-dir=incremental_backup_base_dir]
```

- examples:

```
backup disk-store --dir=/disks/disk_backup1  
--baseline-dir=/disks/disk_backup0/2012-09-10-15-10-12
```

### **compact disk-store**

- availability: online

- arguments:

```
compact disk-store --name=disk_store_name
[--group=membergroup_name(,membergroup_name)*]
```

- examples:

```
compact disk-store --name=Disk1
compact disk-store --name=Disk1 --group=MemberGroup1
```

### **compact offline-disk-store**

- availability: online or offline

- arguments:

```
compact offline-disk-store --name=disk_store_name
--disk-dirs=disk_store_dir(,disk_store_dir)*
[--max-oplog-size=max_size_in_MB] [--J=JVM_args(,JVM_args)*]
```

- examples:

```
compact offline-disk-store --name=Disk1 --disk-dirs=/disks/Disk1
```

### **describe disk-store**

- availability: online

- arguments:

```
describe disk-store --member=member_name --name=disk_store_name
```

- examples:

```
describe disk-store --member=server1 --name=Disk1
```

### **describe offline-disk-store**

- availability: online or offline

- arguments:

```
describe offline-disk-store --name=disk_store_name
--disk-dirs=disk_store_dir(,disk_store_dir)*
[--region=region_name]
```

- examples:

```
describe offline-disk-store --name=Disk1
```

### **list disk-stores**

- availability: online

- arguments: none

- examples:

```
list disk-stores
```

### **revoke missing-disk-store**

- availability: online

- arguments:

```
revoke missing-disk-store --id=disk_store_guid
```

- examples:

```
revoke missing-disk-store --id=-id=60399215-532b-406f-b81f-9b5bd8d1b55a
```

**show missing-disk-store**

- availability: online
- arguments: none
- examples:

```
show missing-disk-stores
```

**upgrade offline-disk-store**

- availability: offline
- arguments:

```
upgrade offline-disk-store --name=disk_store_name  
--disk-dirs=disk_store_dir(,disk_store_dir)*  
[--max-oplog-size=max_size_in_MB] [--J=JVM_args(,JVM_args)*]
```

- examples:

```
upgrade offline-disk-store --name=Disk1  
--disk-dirs=/disks/Disk1,/disks/Disk2
```

**validate offline-disk-store**

- availability: online or offline
- arguments:

```
validate offline-disk-store --name=disk_store_name  
--disk-dirs=disk_store_dir(,disk_store_dir)*
```

- examples:

```
validate offline-disk-store --name=Disk1  
--disk-dirs=/disks/Disk1,/disks/Disk2
```

**Durable CQ and Client Commands****list durable-cqs**

- availability: online
- arguments:

```
list durable-cqs --durable-client-id=durable_client_id  
[--member=member_name] [--group=membergroup_name]
```

- examples:

```
list durable-cqs --durable-client-id=client1
```

**close durable-cq**

- availability: online
- arguments:

```
close durable-cqs --durable-client-id=durable_client_id  
--durable-cq-name=durable_cq_name [--member=member_name]  
[--group=membergroup_name]
```

- examples:

```
close durable-cqs --durable-client-id=client1 --durable-cq-name=cq1
```

**close durable-client**

- availability: online

- arguments:

```
close durable-client --durable-client-id=durable_client_id
[--member=member_name] [--group=membergroup_name]
```

- examples:

```
close durable-client --durable-client-id=client1
```

### **show subscription-queue-size**

- availability: online

- arguments:

```
show subscription-queue-size --durable-client-Id=durable_client_id
[--durable-cq-name=durable_cq_name]
[--member=member_name] [--group=membergroup_name]
```

- examples:

```
show subscription-queue-size --durable-client-Id=client1
--durable-cq-name=cql
```

## **Function Execution Commands**

### **destroy function**

- availability: online

- arguments:

```
destroy function --id=function_id [--group=membergroup_name]
[--member=member_name_or_id]
```

- examples:

```
destroy function --id=Function1
```

### **execute function**

- availability: online

- arguments:

```
execute function --id=function_id [--group=membergroup_name]
[--member=member_name_or_id]
[--region=region_name]
[--arguments=function_arguments(,function_arguments)*]
[--result-collector=class_name_of_result_collector]
[--filter=list_of_keys]
```

- examples:

```
execute function --id=Function1
```

### **list functions**

- availability: online

- arguments:

```
list functions [--matches=java_pattern_to_match]
[--group=membergroup_name(,membergroup_name)*]
[--member=member_name_or_id(,member_name_or_id)*]
```

- examples:

```
list functions
```

## Gateway (WAN) Commands

### list gateways

- availability: online
- arguments:

```
list gateways [ --group=membergroup_name(,membergroup_name)* ]
```

- examples:

```
list gateways
list gateways --group=MemberGroup1
```

### pause gateway-sender

- availability: online
- arguments:

```
pause gateway-sender --id=gateway_sender_id
[ --group=membergroup_name(,membergroup_name)* ] [ --member=member_name ]
```

- examples:

```
pause gateway-sender --id=sender1
```

### resume gateway-sender

- availability: online
- arguments:

```
resume gateway-sender --id=gateway_sender_id
[ --group=membergroup_name(,membergroup_name)* ] [ --member=member_name ]
```

- examples:

```
resume gateway-sender --id=sender1
```

### start gateway-receiver

- availability: online
- arguments:

```
start gateway-receiver [ --group=membergroup_name(,membergroup_name)* ]
[ --member=member_name ]
```

- examples:

```
start gateway-receiver
```

### start gateway-sender

- availability: online
- arguments:

```
start gateway-sender --id=gateway_sender_id
[ --group=membergroup_name(,membergroup_name)* ] [ --member=member_name ]
```

- examples:

```
start gateway-sender --id=sender1
```

**status gateway-receiver**

- availability: online

- arguments:

```
status gateway-receiver [--group=membergroup_name(,membergroup_name)*]
[--member=member_name]
```

- examples:

```
status gateway-receiver
```

**status gateway-sender**

- availability: online

- arguments:

```
status gateway-sender --id=gateway_sender_id
[--group=membergroup_name(,membergroup_name)*] [--member=member_name]
```

- examples:

```
status gateway-sender --id=Gateway1
```

**stop gateway-receiver**

- availability: online

- arguments:

```
stop gateway-receiver [--group=membergroup_name(,membergroup_name)*]
[--member=member_name]
```

- examples:

```
stop gateway-receiver
```

**stop gateway-sender**

- availability: online

- arguments:

```
stop gateway-sender --id=gateway_sender_id
[--group=membergroup_name(,membergroup_name)*] [--member=member_name]
```

- examples:

```
stop gateway-sender --id=Gateway1
```

**GemFire Monitoring Commands****describe member**

- availability: online

- arguments:

```
describe member --name=member_name_or_id
```

- examples:

```
describe member --name=member1
```

**export logs**

- availability: online

- arguments:

```
export logs --dir=target_directory_name  
[--group=membergroup_name(,membergroup_name)*] [--member=member_name_or_id]  
[--log-level=none_or_error_or_info_or_config_or_fine_or_finer_and_finest]  
[--only-log-level=true_or_false] [--merge-log=true_or_false]  
[--start-time=yyyy/MM/dd/MM/mm/ss/SSS/z_OR_yyyy/MM/dd]  
[--end-time=yyyy/MM/dd/MM/mm/ss/SSS/z_OR_yyyy/MM/dd]
```

- examples:

```
export logs --dir=LogDir
```

### **export stack-traces**

- availability: online

- arguments:

```
export stack-traces --file=name_of_export_file [--member=member_name_or_id]  
[--group=membergroup_name]
```

- examples:

```
export stack-traces --file=stack.txt
```

### **gc**

- availability: online

- arguments:

```
gc [--group=membergroup_name(,membergroup_name)*]  
[--member=member_name_or_id]
```

- examples:

```
gc  
gc --member=server1  
gc --group=MemberGroup1
```

### **list members**

- availability: online

- arguments:

```
list members [--group=membergroup_name]
```

- examples:

```
list members  
list members --group=MemberGroup1
```

### **netstat**

- availability: online

- arguments:

```
netstat [--member=member_name_or_id(,member_name_or_id)*]  
[--group=membergroup_name] [--file=text_file_for_output]  
[--with-lsos(=true_or_false_linux/unix_hosts_only)?]
```

- examples:

```
netstat
```

**show dead-locks**

- availability: online
- arguments:

```
show dead-locks --file=text_file_to_write_deadlocks
```

- examples:

```
show dead-locks --file=deadlock1.txt
```

**show log**

- availability: online
- arguments:

```
show log --member=member_name_or_id [--lines=number_of_lines]
```

- examples:

```
show log --member=server1
show log --member=server1 --lines=100
```

**show metrics**

- availability: online
- arguments:

```
show metrics [--member=member_name_or_id] [--region=region_name]
[--file=output_file_name]
[--port=port_number_of_member] [--categories=metric_type(,metric_type)*)
```

- examples:

```
show metrics
show metrics --region=MyRegion
show metrics --categories=cluster,cache
```

**shutdown**

- availability: online
- arguments:

```
shutdown [--time-out=time_in_seconds]
```

- examples:

```
shutdown
shutdown --time-out=60
```

**start jconsole**

- availability: online or offline
- arguments:

```
start jconsole [--interval=refresh_interval_in_seconds]
[--notile(=true_or_false)?] [--pluginpath=dir_or_jar_for_jconsole-plugins]
[--version(=true_or_false)?] [--J=arg_passed_to_jvm(,arg_passed_to_jvm)*]
```

- examples:

```
start jconsole --interval=15 --notile
```

**start jvisualvm**

- availability: online or offline

- arguments:

```
start jvisualvm [--J=arg_passed_to_jvm(,arg_passed_to_jvm)*]
```

- examples:

```
start jvisualvm
```

### **start pulse**

- availability: online or offline

- arguments:

```
start pulse [--url=pulse_http_url]
```

- examples:

```
start pulse
```

```
start pulse --url=http://gemfire.company.com:8080/pulse
```

### **start vsd**

- availability: online or offline

- arguments:

```
start vsd [--file=stats_file_to_open(,stats_file_to_open)*]
```

- examples:

```
start vsd
```

```
start vsd --file=MyStatsFile.gfs
```

## **Index Commands**

### **create index**

- availability: online

- arguments:

```
create index --name=index_name --expression=region_field_to_index  
--region=region_name  
[--member=member_name] [--type=range_or_key_or_hash]  
[--group=membergroup_name]
```

- examples:

```
create index --name=MyIndex --expression=id --region=/exampleRegion  
--type=key
```

### **destroy index**

- availability: online

- arguments:

```
destroy index [--name=index_name] [--region=region_name]  
[--member=member_name] [--group=membergroup_name]
```

- examples:

```
destroy index
```

```
destroy index --name=MyIndex
```

### **list indexes**

- availability: online

- arguments:

```
list indexes [--with-stats(=true_or_false)?]
```

- examples:

```
list indexes
list indexes --with-stats
```

## JMX Connection Commands

### connect

- availability: offline

- arguments:

```
connect [--locator=locator_hostname[port]]
[--jmx-manager=jmx_manager_hostname[port]]
[--user=username] [--password=password]
[--key-store=keystore_filename] [--key-store-password=keystore_password]
[--trust-store=cert_truststore_filename]
[--trust-store-password=cert_truststore_password]
[--ciphers=SSL_TSL_ciphers] [--protocols=SSL_TSL_protocol_versions]
[--security-properties-file=path_to_gfsecurity.properties_file]
```

- examples:

```
connect
connect --locator=MyHost
connect --locator=MyHost[10334]
```

### describe connection

- availability: online or offline



**Note:** Indicates that there is no connection ("Not connected") if executed offline.

- arguments: none

- examples:

```
describe connection
```

### disconnect

- availability: online

- arguments: none

- examples:

```
disconnect
```

## Locator Commands

### start locator

- availability: online or offline

- arguments:

```
start locator --name=locator_name [--dir=locator_working_directory]
[--bind-address=ip_address]
[--force(=true_or_false)?]
```

```
[--hostname-for-clients=hostname_or_ipaddress_for_clients]
[--port=locator_port]
[--properties-file=path_to_gemfire_properties_file]
[--security-properties-file=path_to_gfsecurity_properties_file]
[--initial-heap=init_heap_size_formatted_same_as_-J-Xms_arg]
[--max-heap=max_heap_size_formatted_same_as_-J-Xmx_arg]
[--J=arg_passed_to_jvm(,arg_passed_to_jvm)*]
```

- examples:

```
start locator --name=locator1
```

### **status locator**

- availability: online or offline

- arguments:

```
status locator [--name=locator_name] [--host=ip_address_or_hostname]
[--port=locator_port]
[--pid=locator_pid] [--dir=locator_working_directory]
```

- examples:

```
stop locator
stop locator --name=locator1
stop locator --pid=1032
```

### **stop locator**

- availability: online or offline

- arguments:

```
stop locator [--name=locator_name] [--pid=locator_pid]
[--dir=locator_working_directory]
```

- examples:

```
stop locator
stop locator --name=locator1
stop locator --pid=1032
```

## **Region Commands**

### **create region**

- availability: online

- arguments:

```
create region --name=region_name
[--type=region_type] [--template-region=name_or_path_to_region_to_duplicate]
[--group=membergroup_name(,membergroup_name)*]
[--skip-if-exists=true_or_false]
[--key-constraint=fully_qualified_class_for_region_key_objects]
[--value-constraint=fully_qualified_class_for_entry_value_objects]
[--enable-statistics=true_or_false]
[--entry-idle-time-expiration=time_in_seconds]
[--entry-idle-time-expiration-action=destroy_or_local-destroy_or_invalidate_or_local-invalidate]
[--entry-time-to-live-expiration=time_in_seconds]
[--entry-time-to-live-expiration-action=destroy_or_local-destroy_or_invalidate_or_local-invalidate]
[--region-idle-time-expiration=time_in_seconds]
[--region-idle-time-expiration-action=destroy_or_local-destroy_or_invalidate_or_local-invalidate]
```

```
[--region-time-to-live-expiration=time_in_seconds]
[--region-time-to-live-expiration-action=destroy_or_local_destroy_or_invalidate_or_local-invalidate]

[--disk-store=disk_store_id] [--enable-synchronous-disk=true_or_false]
[--enable-async-conflation=true_or_false]
[--enable-subscription-conflation=true_or_false]
[--cache-listener=fully_qualified_class_for_cache_listener(,another_cache_listener)*]
[--cache-loader=fully_qualified_class_for_cache_loader]
[--cache-writer=fully_qualified_class_for_cache_writer]
[--async-event-queue-id=queue_id(,queue_id)*]
[--gateway-sender-id=queue_id(,queue_id)*]
[--enable-concurrency-checks=true_or_false]
[--enable-cloning=true_or_false]
[--concurrency-level=max_number_of_application_threads]
[--colocated-with=region_name_to_colocate_with]
[--local-max-memory=max_memory_in_MB_for_current_process]
[--recovery-delay=delay_in_milliseconds]
[--redundant-copies=0_or_1_or_2_or_3]
[--startup-recovery-delay=delay_in_milliseconds_for_new_members]
[--total-max-memory=max_memory_in_MB_for_all_processes]
[--total-num-buckets=total_number_of_buckets_for_all_processes]
```

- examples:

```
create region --name=MyRegion --type=PARTITION
```

### **describe region**

- availability: online

- arguments:

```
describe region --name=region_name
```

- examples:

```
describe region --name=MyRegion
```

### **destroy region**

- availability: online

- arguments:

```
destroy region --name=region_name
```

- examples:

```
destroy region --name=MyRegion
```

### **list regions**

- availability: online

- arguments:

```
list regions [--group=membergroup_name] [-member=member_name_or_id]
```

- examples:

```
list regions
list regions --group=MemberGroup1
list regions --member=server1
```

### **rebalance**

- availability: online

- arguments:

```
rebalance
[--include-region=partition_region_path_or_name(,partition_region_path_or_name)*]
[--exclude-region=partition_region_path_or_name(,partition_region_path_or_name)*]
[--time-out=time_to_wait_in_seconds]
[--simulate(=true_or_false)?]
```

- examples:

```
rebalance
rebalance --simulate
```

## Server Commands

### start server

- availability: online or offline

- arguments:

```
start server --name=server_name [--assign-buckets(=true_or_false)?]
[--cache-xml-file=cache_xml_filename_or_path]
[--classpath=location_of_user_classes]
[--disable-default-server(=true_or_false)?]
[--enable-time-statistics(=true_or_false)?] [--force(=true_or_false)?]
[--properties-file=path_to_gemfire.properties_file]
[--security-properties-file=path_to_gfsecurity_properties_file]
[--group=membergroup_name][--license-application-cache=acn_license_serial_number]
[--license-data-management=dmn_license_serial_number]
[--locators=locator_hostname[port]]
[--log-level=finest_or_finer_or_fine_or_config_or_info_or_warning_or_severe_or_none]

[--mcast-address=mcast_ip_address]
[--mcast-port=mcast_port_or_0_to_disable_mcast]
[--memcached-port=port_number_of_gemcached_server]
[--memcached-protocol=ascii_or_binary]
[--rebalance(=true_or_false)?] [--server-bind-address=ip_address]
[--server-port=cache_server_port_for_client_connections]
[--statistic-archive-file=file_to_write_statistic_samples]
[--dir=server_working_directory]
[--initial-heap=init_heap_size_formatted_same_as_-J-Xms_arg]
[--max-heap=max_heap_size_formatted_same_as_-J-Xmx_arg]
[--J=arg_passed_to_jvm(,arg_passed_to_jvm)*]
```

- examples:

```
start server --name=server1
```

### status server

- availability: online or offline

- arguments:

```
status server [--name=server_name] [--pid=server_pid]
[--dir=server_working_directory]
```

- examples:

```
status server
status server --name=server1
```

**stop server**

- availability: online or offline
- arguments:

```
stop server [--name=server_name] [--pid=server_pid]
[--dir=server_working_directory]
```

- examples:

```
stop server --name=server1
```

**gfsh Command Help**

This section provides help and usage information on all gfsh commands, listed alphabetically.

**alter**

Modify an existing GemFire resource.

**alter disk-store**

Modify certain configuration options for a specified region or remove a region from an offline disk-store. You can also use the command to reset specific configuration attributes of the disk-store to their default values. For example, you could use `alter disk-store` command to set the region's LRU action to `overflow-to-disk`.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
alter disk-store --name=value --region=value --disk-dirs=value(,value)*
[--lru-algorithm=value] [--lru-action=value] [--lru-limit=value]
[--concurrency-level=value] [--initial-capacity=value] [--load-factor=value]
[--enable-statistics=value] [--remove(=value)?]
```

<code>name</code>	Required. Name of the disk-store whose contents will be altered.
<code>region</code>	Required. Name/Path of the region in the disk store to alter.
<code>disk-dirs</code>	Required. Directories where data for the disk store was previously written.
<code>lru-algorithm</code>	Least recently used eviction algorithm. Valid types are: <code>none</code> , <code>lru-entry-count</code> , <code>lru-heap-percentage</code> , or <code>lru-memory-size</code>
<code>lru-action</code>	Action to take when evicting entries from the region. Valid values are: <code>none</code> , <code>overflow-to-disk</code> and <code>local-destroy</code> .
<code>lru-limit</code>	Number of entries allowed in the region before eviction will occur.
<code>concurrency-level</code>	An estimate of the maximum number of application threads that will concurrently access a region entry at one time. This attribute does not apply to partitioned regions.
<code>initial-capacity</code>	Together with <code>--load-factor</code> , sets the parameters on the underlying <code>java.util.ConcurrentHashMap</code> used for storing region entries.
<code>load-factor</code>	Together with <code>--initial-capacity</code> , sets the parameters on the underlying <code>java.util.ConcurrentHashMap</code> used for storing region entries. This must be a floating point number between 0 and 1, inclusive.
<code>enable-statistics</code>	Sets the region's statistics enabled. Valid values are true or false. Default is false.
<code>remove</code>	Whether to remove the region from the disk-store. Default is false. If a value is not specified for this parameter, the default value of true is used.

**Example Commands:**

```
alter disk-store --name=DiskStore1 --region=region1
--disk-dirs=/Disks/DiskStore1
alter disk-store --name=DiskStore1 --region=region1
--disk-dirs=/Disks/DiskStore1 --remove
```

**alter runtime**

Alter a subset of configuration properties for specific member or members while the member or members are running.

For definitions of these configuration properties, see [gemfire.properties and gfsecurity.properties \(GemFire Properties\)](#) on page 851.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
alter runtime [--member=value] [--group=value]
[--archive-disk-space-limit=value] [--archive-file-size-limit=value]
[--log-disk-space-limit=value] [--log-file-size-limit=value]
[--log-level=value]
[--statistic-archive-file=value] [--statistic-sample-rate=value]
[--enable-statistics=value]
```

member	Required if --group is not specified. Name/Id of the member whose configuration is to be altered at runtime.
group	Required if --member is not specified. Name of the group whose member's runtime configuration is to be altered.
archive-disk-space-limit	Archive disk space limit. Valid values are (in megabytes): 0 - 1000000.
archive-file-size-limit	Archive file size limit. Valid values are (in megabytes): 0 - 1000000.
log-disk-space-limit	Log disk space limit. Valid values are (in megabytes): 0 - 1000000.
log-file-size-limit	Log file size limit. Valid values are (in megabytes): 0 - 1000000.
log-level	Log level. Valid values are: none, error, info, config, fine, finer and finest.
statistic-archive-file	File to which the statistics will be written.
statistic-sample-rate	Statistic sampling rate. Valid values are (in milliseconds): 100 - 60000.
enable-statistics	Whether statistic sampling should be enabled. Valid values are: true and false.

**Example Commands:**

```
alter runtime --member=server1 --log-level=fine --enable-statistics=true
```

**Sample Output:**

```
gfsh>alter runtime --member=server1 --log-level=fine
--enable-statistics=true
Runtime configuration altered successfully for the following member(s)
192.168.129.129(server1:240)<v1>:64871
```

**backup disk-store**

Back up persistent data from all members to the specified directory.

The specified directory must exist on all members, but it can be a local directory on each machine. This command ensures that backup files are not corrupted by concurrent operations. Backing up a running system using the file system's copy command is not recommended.

You can also use this command to perform an incremental backup. See [Back Up and Restore a Disk Store](#) on page 478 for more information on incremental backup.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

#### Syntax:

```
backup disk-store --dir=value [--baseline-dir=value]
```

dir	Required. Directory to which backup files are written.
baseline-dir	<p>Directory that contains the baseline backup used for comparison during an incremental backup.</p> <p>An incremental backup operation backs up any data that is not present in the directory specified in --baseline-dir. If the member cannot find previously backed up data or if the previously backed up data is corrupt, the command performs a full backup on that member.</p>

#### Example Commands:

```
backup disk-store --dir=data/backups
backup disk-store --dir=data/backup/disk-store
--baselineDir=data/backups/2012-09-24-17-08-50
```

#### Sample Output:

```
gfsh>backup disk-store --dir=data/backups
The following disk stores were backed up successfully

Member | UUID | Directory
| Host |
-----+-----+-----+
-----+-----+
server2 | a6bb11f0-0baa-45c9-b23e-64876d02a586 | c:\vFabricGemFire70\Latest\serv
er2\. | 192.168.129.129
server1 | 8dc365bd-c086-4af4-99d0-86b0b521aa04 | c:\vFabricGemFire70\Latest\serv
er1\. | 192.168.129.129
```

#### close

Close durable client CQs and durable clients.

#### close durable-client

Attempts to close the durable client. The client must be disconnected for this command to work.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

#### Syntax:

```
close durable-client --durable-client-id=value [--member=value]
[--group=value]
```

durable-client-id	Required. The ID of the durable client.
member	Name/Id of the member for which the durable client is to be closed.
group	Group of members for which the durable client is to be closed.

**Example Commands:**

```
close durable-client --durable-client-id=client1
```

**Sample Output:**

```
gfsh>close durable-client --durable-client-id=client1
Closed the durable client : "client1". on following members.
1.server4
2.server3
```

**Error Messages:**

```
gfsh>close durable-cq --durable-cq-name=cq1 --durable-client-id=client1
Could not close the durable-cq : "cq1" for the durable-client-id : "client1"
due to following reasons.

CacheClientProxy: Could not drain cq cq1 because client proxy id client1
is connected.
Occurred on members
1.server4
2.server3

No client found with client-id : client1
Occurred on members
1.server1
```

**close durable-cq**

Closes the durable CQ registered by the durable client and drain events held for the durable CQ from the subscription queue.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
close durable-cq --durable-client-id=value --durable-cq-name=value
[--member=value] [--group=value]
```

durable-client-id	Required. The ID of the durable client.
durable-cq-name	Required. Name of the CQ to be closed.
member	Name/Id of the member for which the durable client is registered and the durable CQ to be closed.
group	Group of members for which the durable client is registered and the durable CQ to be closed.

**Example Commands:**

```
close durable-cq --durable-client-id=client1 --durable-cq-name=cq1
```

**Sample Output:**

```
gfsh>close durable-cq --durable-cq-name=cq1 --durable-client-id=client1
Closed the durable cq : "cq1" for the durable client : "client1". on
following members.
1.server4
2.server3
```

**Error Messages:**

```
gfsh>close durable-client --durable-client-id=client1
Unable to close the durable client : "client1" due to following reasons.
```

```
Cannot close a running durable client : client1
Occurred on members
1.server4
2.server3

No client found with client-id : client1
Occurred on members
1.server1
```

**compact**

Compact online and offline disk-stores.

**compact disk-store**

Compact a disk store on all members with that disk store. This command uses the compaction threshold that each member has configured for its disk stores. The disk store must have the "allow-force-compaction" property set to true.

See [Running Compaction on Disk Store Log Files](#) on page 473 for more information.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
compact disk-store --name=value [--group=value(,value)*]
```

name	Required. Name of the disk store to be compacted.
group	Group(s) of members that will perform disk compaction. If no group is specified, then the disk store will be compacted by all members.

**Example Commands:**

```
compact disk-store --name=Disk1
compact disk-store --name=Disk1 --group=MemberGroup1,MemberGroup2
```

**Error Messages:**

```
"Disk store \"\{0}\\" does not exist.";
" for group(s) \"\{0}\\"";
"No members found in the specified group(s) \"\{0}\\".";
"Compaction was attempted but nothing to compact.";
"Error occurred while doing compaction. Reason: \"\{0}\\"";
```

**compact offline-disk-store**

Compact an offline disk store. If the disk store is large, additional memory may need to be allocated to the process using the --J=-Xmx??? parameter.

See [Running Compaction on Disk Store Log Files](#) on page 473 for more information.



**Note:** Do not perform offline compaction on the baseline directory of an incremental backup.

**Availability:** Online or offline.

**Syntax:**

```
compact offline-disk-store --name=value --disk-dirs=value(,value)*
[--max-oplog-size=value] [--J=value(,value)*]
```

name	Required. Name of the offline disk store to be compacted.
disk-dirs	Required. Directories where data for the disk store was previously written.
max-oplog-size	Maximum size (in megabytes) of the oplogs created by compaction.

J

Arguments passed to the Java Virtual Machine performing the compact operation on the disk store.

**Example Commands:**

```
compact offline-disk-store --name=Disk2 --disk-dirs=/Disks/Disk2
compact offline-disk-store --name=Disk2 --disk-dirs=/Disks/Disk2
--max-oplog-size=512 -J=-Xmx1024m
```

**connect**

Connect to a jmx-manager either directly or via a locator.

If connecting via a locator, and a jmx-manager does not already exist, the locator will start one.

gfsh connects as a discovery client to the locator service and asks where the JMX Manager is. Locator knows that there is no member currently configured as the JMX manager and simply starts up the JMX manager service within itself. gfsh connects as a JMX client to locator JMX RMI port.

**Availability:** Offline. You will receive a notification "Already connected to: host[port]" if you are already connected.

**Syntax:**

```
connect [--locator=value] [--jmx-manager=value] [--user=value]
[--password=value] [--key-store=value] [--key-store-password=value]
[--trust-store=value] [--trust-store-password=value] [--ciphers=value]
[--protocols=value] [--security-properties-file=value] [--use-ssl=(value)?]
```

locator	Network address of the Locator in the form: host[port]. The default if the parameter is not specified: <b>localhost[10334]</b>
jmx-manager	Network address of the jmx-manager in the form: host[port].
user	User name to securely connect to the jmx-manager. If the <b>--password</b> parameter is not specified then it will be prompted for.
password	Password to be used as credential which is required to connect to a jmx-manager securely.
key-store	Java keystore file containing this application's certificate and private key. If the <b>--key-store-password</b> parameter is not specified then it will be prompted for.
key-store-password	Password to access the private key from the keystore file specified by <b>--key-store</b> .
trust-store	Java keystore file containing the collection of CA certificates trusted by this application. If the <b>--trust-store-password</b> parameter is not specified then it will be prompted for.
trust-store-password	Password to unlock the keystore file specified by <b>--trust-store</b> .
ciphers	SSL/TLS ciphers used when encrypting the connection. The default is "any".
ssl-protocols	SSL/TLS protocol versions to enable when encrypting the connection. The default is "any".
security-properties-file	The <b>gfsecurity.properties</b> file for configuring gfsh to connect to the Locator/Manager. The file's path can be absolute or relative to the current gfsh directory.
use-ssl	Whether to use SSL for communication with Locator and/or JMX Manager. If set to "true", the connect command will also read <b>gfsecurity.properties</b> . SSL Options take precedence over properties file. If none are specified, defaults will be used. The default value for this options is "false". If the parameter is specified without a value, the default value is true. Default value if the parameter is not specified is false.

**Example Commands:**

If you do not specify a locator or jmx-manager, gfsh connects to the locator on the localhost at the default port.

```
gfsh>connect
```

#### Sample Output:

```
gfsh>connect
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=GemFireStymon, port=1099] ..
Successfully connected to: [host=GemFireStymon, port=1099]
```

#### Error Messages:

```
"Locator could not find a JMX Manager";
"jmx password must be specified.";
"Could not connect to : {0}. {1}";
"Could not find a GemFire jmx-manager service running at {0}.";
"Could not connect to GemFire Locator service at {0}."
```

#### **create**

Create gateway receivers, indexes, and regions.

#### **create index**

Create an index that can be used when executing queries.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

#### Syntax:

```
create index --name=value --expression=value --region=value
[--member=value] [--type=value] [--group=value]
```

name	Required. Name for the index to create.
expression	Required. Field of the region values that are referenced by the index.
region	Required. Name/Path of the region which corresponds to the "from" clause in a query.
member	Name/Id of the member on which index will be created.
type	Type of the index. Valid values are: range, key and hash. The default type is range if the parameter is not specified.
group	The index will be created on all the members in this member group.

#### Example Commands:

```
create index --name=MyKeyIndex --expression=region1.Id --region=region1
```

#### **create region**

Create a region with given path and configuration. You must specify either a --type or a --template-region for initial configuration when creating a region. Specifying a --key-constraint and --value-constraint makes object type information available during querying and indexing.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

#### Syntax:

```
create region --name=value [--type=value] [--template-region=value]
[--group=value[,value]*] [--skip-if-exists=value] [--key-constraint=value]
[--value-constraint=value] [--enable-statistics=value]
```

```

[--entry-idle-time-expiration=value]
[--entry-idle-time-expiration-action=value]
[--entry-time-to-live-expiration=value]
[--entry-time-to-live-expiration-action=value]
[--region-idle-time-expiration=value]
[--region-idle-time-expiration-action=value]
[--region-time-to-live-expiration=value]
[--region-time-to-live-expiration-action=value]
[--disk-store=value] [--enable-synchronous-disk=value]
[--enable-async-conflation=value]
[--enable-subscription-conflation=value] [--cache-listener=value(,value)*]

[--cache-loader=value] [--cache-writer=value]
[--async-event-queue-id=value(,value)*]
[--gateway-sender-id=value(,value)*] [--enable-concurrency-checks=value]
[--enable-cloning=value] [--concurrency-level=value] [--colocated-with=value]

[--local-max-memory=value] [--recovery-delay=value]
[--redundant-copies=value]
[--startup-recovery-delay=value] [--total-max-memory=value]
[--total-num-buckets=value]

```

<b>name</b>	Required. Name/Path of the region to be created.
<b>type</b>	Required (if template-region is not specified.) Type of region to create. Options include: PARTITION, PARTITION_REDUNDANT, REPLICATE, LOCAL, etc.  To get a list of all region type options, add the <code>--type</code> parameter and then select the TAB key to display a full list.
<b>template-region</b>	Required (if type is not specified.) Name/Path of the region whose attributes should be duplicated when creating this region.
<b>group</b>	Group(s) of members on which the region will be created.
<b>skip-if-exists</b>	Skip region creation if the region already exists. If the parameter is specified without a value, the value is set to true. If the parameter is not specified, the value (default) is false.
<b>key-constraint</b>	Fully qualified class name of the objects allowed as region keys. Ensures that keys for region entries are all of the same class.
<b>value-constraint</b>	Fully qualified class name of the objects allowed as region values. If not specified, then region values can be of any class.
<b>enable-statistics</b>	Whether to gather statistics for the region. Must be true to use expiration on the region.
<b>entry-idle-time-expiration</b>	How long the region's entries can remain in the cache without being accessed. The default is no expiration of this type.
<b>entry-idle-time-expiration-action</b>	Action to be taken on an entry that has exceeded the idle expiration. Valid expiration actions include destroy, local-destroy, invalidate (default), local-invalidate.
<b>entry-time-to-live-expiration</b>	How long the region's entries can remain in the cache without being accessed or updated. The default is no expiration of this type.
<b>entry-time-to-live-expiration-action</b>	Action to be taken on an entry that has exceeded the TTL expiration. Valid expiration actions include destroy, local-destroy, invalidate (default), local-invalidate.
<b>region-idle-time-expiration</b>	How long the region can remain in the cache without being accessed. The default is no expiration of this type.
<b>region-idle-time-expiration-action</b>	Action to be taken on a region that has exceeded the idle expiration. Valid expiration actions include destroy, local-destroy, invalidate (default), local-invalidate.
<b>region-time-to-live-expiration</b>	How long the region can remain in the cache without being accessed or updated. The default is no expiration of this type.
<b>region-time-to-live-expiration-action</b>	Action to be taken on a region that has exceeded the TTL expiration. Valid expiration actions include destroy, local-destroy, invalidate (default), local-invalidate.

disk-store	Disk Store to be used by this region. The <a href="#">list disk-stores</a> on page 697 command can be used to display existing disk stores.
enable-synchronous-disk	Whether writes are done synchronously for regions that persist data to disk.
enable-async-conflation	Whether to allow aggregation of asynchronous TCP/IP messages sent by the producer member of the region. A false value causes all asynchronous messages to be sent individually.
enable-subscription-conflation	Whether the server should conflate its messages to the client. A false value causes all server-client messages to be sent individually.
cache-listener	Fully qualified class name of a plug-in to be instantiated for receiving after-event notification of changes to the region and its entries. Any number of cache listeners can be configured.
cache-loader	Fully qualified class name of a plug-in to be instantiated for receiving notification of cache misses in the region. At most, one cache loader can be defined in each member for the region. For distributed regions, a cache loader may be invoked remotely from other members that have the region defined.
cache-writer	Fully qualified class name of a plug-in to be instantiated for receiving before-event notification of changes to the region and its entries. The plug-in may cancel the event. At most, one cache writer can be defined in each member for the region.
async-event-queue-id	IDs of the Async Event Queues that will be used for write-behind operations.
gateway-sender-id	IDs of the Gateway Senders to which data will be routed.
enable-concurrency-checks	Whether Region Version Vectors are implemented. Region Version Vectors are an extension to the versioning scheme that aid in synchronization of replicated regions.
enable-cloning	Determines how fromDelta applies deltas to the local cache for delta propagation. When true, the updates are applied to a clone of the value and then the clone is saved to the cache. When false, the value is modified in place in the cache.
concurrency-level	Estimate of the maximum number of application threads that will concurrently access a region entry at one time. This attribute does not apply to partitioned regions.
colocated-with	Central Region with which this region should be colocated.
local-max-memory	Maximum amount of memory, in megabytes, to be used by the region in this process. (The default is 90% of available heap.)
recovery-delay	Delay in milliseconds that existing members will wait before satisfying redundancy after another member crashes. The default (value of -1) indicates that redundancy will not be recovered after a failure.
redundant-copies	Number of extra copies of buckets desired. Extra copies allow for both high availability in the face of VM departure (intended or unintended) and load balancing read operations. (Allowed values: 0, 1, 2 and 3)
startup-recovery-delay	Delay in milliseconds that new members will wait before satisfying redundancy. -1 indicates that adding new members will not trigger redundancy recovery. The default is to recover redundancy immediately when a new member is added.
total-max-memory	Maximum amount of memory, in megabytes, to be used by the region in all processes.
total-num-buckets	Total number of hash buckets to be used by the region in all processes. (Default: 113).

### Example Commands:

```
create region --name=region1 --type=REPLICATE_PERSISTENT \
--cache-writer=com.gemstone.gemfire.examples.MyCacheWriter \
--group=Group1 --disk-store=DiskStore1

create region --name=region12 --template-region=/region1

create region --name=region2 --type=REPLICATE \
--cache-listener=com.gemstone.gemfire.examples.MyCacheListener1, \
com.gemstone.gemfire.examples.MyCacheListener2 \
--group=Group1,Group2
```

```

create region --name=region3 --type=PARTITION_PERSISTENT --redundant-copies=2
--total-max-memory=1000 \
--startup-recovery-delay=5 --total-num-buckets=100 --disk-store=DiskStore2
\
--cache-listener=com.gemstone.gemfire.examples.MyCacheListener3 \
--group=Group2 --gateway-sender-id=2,3

create region --name=region4 --type=REPLICATE_PROXY
--cache-listener=com.gemstone.gemfire.examples.MyCacheListener1
--group=Group1,Group2

```

**Sample Output:**

```

gfsh>create region --name=myRegion --type=LOCAL
Member | Status
----- | -----
server1 | Region "/myRegion" created on "server1"

```

**debug**

Enable or disable debugging output in gfsh.

**Availability:** Online or offline.

**Syntax:**

```
debug --state=value
```

state	Whether to turn debugging ON or OFF. Valid options are: ON, OFF (Case insensitive) Debugging is OFF by default.
-------	---

**Example Commands:**

```

debug --state=off
debug --state=on

```

**Sample Output:**

```

gfsh>debug --state=on
Debug is on

```

**deploy**

Deploy JAR-packaged applications to a member or members.

Only one of either --jar or --dir may be specified.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
deploy [--group=value(,value)*] [--jar=value] [--dir=value]
```

group	Group(s) to which the specified JARs will be deployed. If this option is not specified, the deployment will occur on all members.
jar	Path of the JAR to deploy.
dir	Directory from which to deploy the JARs.

**Example Commands:**

```
deploy --jar=group1_functions.jar --group=Group1
```

```
deploy --dir=libs/group1-libs --group=Group2
```

**Sample Output:**

```
gfsh> deploy --jar=group1_functions.jar --group=Group1
Member | Deployed JAR | Deployed JAR Location
----- | ----- |
-----
datanode1 | group1_functions.jar | /usr/local/gemfire/deploy/GF#group1_functions.jar#1
datanode2 | group1_functions.jar | /usr/local/gemfire/deploy/GF#group1_functions.jar#1

gfsh> deploy --dir=libs/group1-libs --group=Group2
Deploying files: group2_functions.jar, group2_dependencies.jar
Total file size is: 0.64MB

Continue? (Y/n): Y

Member | Deployed JAR | Deployed JAR Location
----- | ----- |
-----
datanode3 | group2_functions.jar | /usr/local/gemfire/deploy/GF#group2_functions.jar#1
datanode3 | group2_dependencies.jar | /usr/local/gemfire/deploy/GF#group2_dependencies.jar#1
datanode4 | group2_functions.jar | /usr/local/gemfire/deploy/GF#group2_functions.jar#1
datanode4 | group2_dependencies.jar | /usr/local/gemfire/deploy/GF#group2_dependencies.jar#1
```

**describe**

Display details of a member's configuration, shell connection, disk-stores, members, or regions.

**describe config**

Display the configuration of a member.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
describe config --member=value [--hide-defaults(=value)?]
```

member	Name or ID of a member whose configuration is to be shown.
hide-defaults	Whether to hide configuration information for properties with the default value. If the value for this parameter is not specified, the value is assumed to be true. If the parameter is not specified, the default is true.

**Example Commands:**

```
describe config --member=Member1;
```

**Sample Output:**

```
gfsh>describe config --member=server1
Configuration of member : "server1"
```

JVM command line arguments

```

-----
-Dgemfire.mcast-port=0
-Dgemfire.locators=localhost[10334]

GemFire properties defined using the API
.....  

log-file : vf.gf.server.log  

name : server1

GemFire properties defined at the runtime
.....  

log-level : finest  

statistic-sampling-enabled : true

Cache attributes
.....  

is-server : true

Cache-server attributes
. bind-address : localhost

```

**describe connection**

Display connection information details.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
describe connection
```

**Example Commands:**

```
describe connection
```

**Sample Output:**

```
gfsh>describe connection
Connection Endpoints
-----
GemFireSty whole [1099]
```

**describe disk-store**

Display information about a member's disk store.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
describe disk-store --member=value --name=value
```

member	Required. Name/Id of the member with the disk store to be described.
name	Required. Name for the disk store to be described.

**Example Commands:**

```
describe disk-store --member=server1 --name=DiskStore1
```

**Sample Output:**

```
gfsh>describe disk-store --member=server1 --name=DiskStore1
Member Name: : server1
Allow Force Compaction: : No
```

```

Auto Compaction: : Yes
Compaction Threshold: : 50
Max Oplog Size: : 1024
Queue Size: : 0
Time Interval: : 1000
Write Buffer Size: : 32768
PDX Serialization Meta-Data Stored: : No

```

Disk Directory	Size
/export/data/gemfire/vml/diskDir/disk-1	2147483647

Region Path	Region Name	Persistent	Overflow To Disk
/producer	producer	Yes	No
/producer-factory	producer-factory	Yes	No

Bind Address	Hostname for Clients	Port
10.127.244.5	serverOne.domain.com	10255

Gateway ID	Persistent
12345	Yes
67890	No

Async Event Queue ID
Q1
Qn

## describe member

Display details of a member with given name/id.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

### Syntax:

```
describe member --name=value
```

name	Display information about a member, including name, id, groups, regions, etc.
------	---

### Example Commands:

```
describe member --name=server1
```

**Sample Output:**

```
gfsh>describe member --name=server1
Name      : server1
Id        : GemFireStyMon(server1:240)<v1>:64871
Host      : 192.168.129.129
Regions   : region4
region5
region3
region2
region1

PID       : 240
Groups    :
Used Heap : 5M
Max Heap  : 123M
Working Dir: c:\vFabricGemFire70\Latest\server1
Log file  : C:\vFabricGemFire70\Latest\server1\vf.gf.server.log
Locators   : localhost[10334]

Server Bind      : localhost
Server Port      : 40404
Running          : true
Client Connections : 0
```

**describe region**

Display the attributes and key information of a region.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
describe region --name=value
```

name	Required. Name/Path of the region to be described.
------	--

**Example Commands:**

```
describe region --name=region1
```

**Sample Output:**

```
gfsh>describe region --name=region1
.
.
.
Name      : region1
Data Policy : normal
Hosting Members : server1
server2

Non-Default Attributes Shared By Hosting Members

Type    |    Name    |    Value
----- | ----- | -----
Region  | data-policy | NORMAL
          | size        | 0
```

**destroy**

Delete or unregister functions, remove indexes, and remove regions.

**destroy function**

Destroy or unregister a function. The default is for the function to be unregistered from all members.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
destroy function --id=value [--group=value] [--member=value]
```

id	Required. Unique function identifier. Use the list functions command to obtain the ID.
group	Group of members from which this function will be unregistered.
member	Name/Id of the member from which this function will be unregistered.

**Example Commands:**

```
(1) destroy function --id=InterestCalculations
(2) destroy function --id=InterestCalculations --member=server1
(3) destroy function --id=InterestCalculations --group=Group1
```

**destroy index**

Destroy or remove the specified index.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
destroy index [--name=value] [--region=value] [--member=value]
[--group=value]
```



**Note:** You must specify at least one of the parameter options. If you enter `destroy index` without any parameters, the command will ask you to specify at least one option.

name	Name for the index to be removed.
member	Id of the member on which index is to be created.
region	Name of the region , from where an index or all indexes are to be destroyed.
group	The index will be created on all the members in this member group.

**Example Commands:**

```
destroy index --member=server2
destroy index --name=MyKeyIndex
```

**destroy region**

Destroy or remove a region.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
destroy region --name=value
```

name	Name/Path of the region to be removed.
------	--

**Example Commands:**

```
destroy region --name=region4
destroy region --name=/region1/subregion1
```

**Sample Output:**

```
gfsh>destroy region --name=region1
"region1" destroyed successfully.
```

**disconnect**

Close any active connection(s).

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
disconnect
```

**Example Commands:**

```
disconnect
```

**Sample Output:**

```
gfsh>disconnect
Disconnecting from: GemFireStymon[1099]
Disconnected from : GemFireStymon[1099]
```

**Error Messages:**

```
Error occurred while disconnecting: {0}
```

Not connected!

**echo**

Echo the given text, which may include system and user variables.

The command can also echo gfsh environment properties (using '[set variable](#) on page 708' command) if variable name is pre-pended with '\$' - like UNIX.

See [Useful gfsh Shell Variables](#) on page 654 for a list of gfsh environment variables.

**Availability:** Online or offline.

**Syntax:**

```
echo [--string=value]
```

string	String to be echoed. For example, "SYS_USER variable is set to \${SYS_USER}".
--------	---

**Example Commands:**

```
echo --string="Hello World!"
echo --string="Hello World! This is ${SYS_USER}"
echo --string=${APP_FETCH_SIZE}

//To see all the variable set in the shell:
echo --string=$*
```

**Sample Output:**

Property	Value
APP_FETCH_SIZE	1000
APP_LAST_EXIT_STATUS	0
APP_LOG_FILE	/home/stymon/gfsh-2013-03-04_12-13-44.log
APP_NAME	gfsh
APP_PWD	/home/stymon
APP_QUERY_RESULTS_DISPLAY_MODE	table
APP QUIET_EXECUTION	false
SYS_CLASSPATH	[REDACTED]
SYS_GEMFIRE_DIR	/home/stymon/vFabric_GemFire_701_b40169
SYS_HOST_NAME	stymon
SYS_JAVA_VERSION	1.6.0_38
SYS_OS	Linux
SYS_OS_LINE_SEPERATOR	[REDACTED]
SYS_USER	stymon
SYS_USER_HOME	/home/stymon

**encrypt**

Encrypt a password for later use.

**encrypt password**

Encrypt a password for use in data source configuration. See [Configuring Database Connections Using JNDI](#) on page 404 and [Encrypting Passwords for Use in cache.xml](#) on page 505 for more information on how to use these encrypted passwords.

**Availability:** Online or offline.

**Syntax:**

```
encrypt password --password=value
```

password	Required. Password string to be encrypted.
----------	--

**Example Commands:**

```
encrypt password --password=Aht23fb1234#q
```

**Sample Output:**

```
gfsh>encrypt password --password=Aht23fb1234#q
21793243D4F4B1C90B032D0CC0A80821
```

**execute**

Execute functions on members or regions.

**execute function**

Execute the function with the specified ID. By default will execute on all members.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
execute function --id=value [--group=value] [--member=value] [--region=value]
[--arguments=value(,value)*] [--result-collector=value]
[--filter=value]
```

<b>id</b>	Required. ID of the function to execute.
<b>group</b>	Group of members on which this function should be executed. Note that you can only supply one member group.
<b>member</b>	Name/Id of the member on which the function will be executed.
<b>region</b>	Region on which the data dependent function will be executed.
<b>arguments</b>	Arguments to the function in comma separated String format.
<b>result-collector</b>	Fully qualified class name of the ResultCollector to instantiate for gathering results.
<b>filter</b>	Key list which causes the function to only be executed on members which have entries with these keys.

**Example Commands:**

```
execute function --id=InterestCalculations --region=/InterestRegion
execute function --id=InterestCalculations --member=server1
execute function --id=InterestCalculations --group=Group1
```

**exit**

Exit the gfsh shell. You can also use `quit` to exit the shell.

Exits the gfsh shell and returns to the OS shell.

**Availability:** Online or offline.

**Syntax:**

```
exit
```

**Example Commands:**

```
exit
```

**export**

Export configurations, data, logs and stack-traces.

**export config**

Export configuration properties for a member or members. If you do not specify any parameters, all member configuration will be exported.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
export config [--member=value(,value)*] [--group=value(,value)*]
[--dir=value]
```

<b>member</b>	Name/Id of the member(s) whose configuration will be exported.
<b>group</b>	Group(s) of members whose configuration will be exported.
<b>dir</b>	Directory to which the exported configuration files will be written.

**Example Commands:**

```
export config
export config --member=member1
```

**Sample Output:**

```
gfsh>export config --member=member1
Downloading Cache XML file: c:\vFabricGemFire70\Latest\.\\member1-cache.xml
Downloading properties file:
c:\\vFabricGemFire70\\Latest\\.\\member1-gf.properties
```

**export data**

Export user data from a region to a file.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
export data --region=value --file=value --member=value
```

region	Required. Region from which data is to be exported.
file	Required. File to which the exported data is to be written. The file must have an extension of ".gfd".
member	Required. Name/Id of a member that hosts the region. The data will be exported to the specified file on the host where the member is running.

**Example Commands:**

```
export data --region=region2 --file=region2_20121001.gfd --member=server2
```

**Sample Output:**

```
gfsh>export data --region=region2 --file=region2_20121001.gfd
--member=server1
Data successfully exported from region : region2 to file :
C:\\vFabricGemFire70\\
Latest\\server1\\region2_20121001.gfd on host : 192.168.129.129
```

**export logs**

Export/dump logs to a given directory.

All files that have logs in the specified time range will be exported. If no time range is specified, all logs will be exported.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
export logs --dir=value [--group=value(,value)*] [--member=value] [--log-level=value]
[--only-log-level=value] [--merge-log=value]
[--start-time=value]
[--end-time=value]
```

dir	Required. Directory to which log files will be written.
group	Group of members whose log files will be exported.
member	Name/Id of the member whose log files will be exported.

<code>log-level</code>	Minimum level of log entries to export. Valid values are: <code>none</code> , <code>error</code> , <code>info</code> , <code>config</code> , <code>fine</code> , <code>finer</code> and <code>finest</code> . The default is "info".
<code>only-log-level</code>	Whether to only include those entries that exactly match the <code>--log-level</code> specified.
<code>merge-log</code>	Whether to merge logs after exporting to the target directory.
<code>start-time</code>	Log entries that occurred after this time will be exported. The default is no limit. Format: <code>yyyy/MM/dd HH:mm:ss/SSS/z</code> OR <code>yyyy/MM/dd</code>
<code>end-time</code>	Log entries that occurred before this time will be exported. The default is no limit. Format: <code>yyyy/MM/dd HH:mm:ss/SSS/z</code> OR <code>yyyy/MM/dd</code>

**Example Commands:**

```
export logs --dir=data/logs
```

**Sample Output:**

```
gfsh>export logs --dir=data/logs
Successfully exported to directory data/logs
```

**export stack-traces**

Export the stack trace for a member or members.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
export stack-traces --file=value [--member=value] [--group=value]
```

<code>file</code>	Required. Filename to which the stack-traces will be written.
<code>member</code>	Name/Id of the member whose log files will be exported.
<code>group</code>	Group of members whose log files will be exported.

**Example Commands:**

```
export stack-traces --file=stack.txt
```

**Sample Output:**

```
gfsh>export stack-traces --file=stack.txt
stack-trace(s) exported to file:
C:\vFabricGemFire70\Latest\locator1\stack.txt
On host : GemFireStymon
```

**gc**

Force GC (Garbage Collection) on a member or members.

The default is for garbage collection to occur on all caching members.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
gc [--group=value(,value)*] [--member=value]
```

<code>group</code>	Group(s) of members on which garbage collection will be done.
<code>member</code>	Name/Id of the member on which garbage collection will be done.

**Example Commands:**

```
gc --member=server1
gc --group=Group1
gc
```

**Sample Output:**

```
gfsh>gc
Successfully executed GC
```

**get**

Display an entry in a region.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
get --key=value --region=value [--key-class=value] [--value-class=value]
```

key	Required. String or JSON text from which to create the key. Examples include: "James", "100L" and {"id": "134s"}.
region	Required. Region from which to get the entry.
key-class	Fully qualified class name of the key's type. The default is the key constraint for the current region or String.
value-class	Fully qualified class name of the value's type. The default is the value constraint for the current region or String.

**Example Commands:**

```
get --key=('id':'133abg124') --region=region1
// Retrieving when key type is a wrapper(primitive)/String
get --key='133abg124' --region=/region1/region12
--value-class=data.ProfileDetails

get --key='100L' --region=/region1/region12
--value-class=data.ProfileDetails
--key-class=java.lang.Long
```

**Sample Output:**

```
gfsh>get --key='123' --region=region1
Result      : true
Key Class   : java.lang.String
Key         : ('123')
Value Class : java.lang.String
Value       : ABC
```

**help**

Display syntax and usage information for all the available commands.

Typing help without a command as an argument lists all available commands.

**Availability:** Online or offline.

**Syntax:**

```
help [command]
```

command	Name of the command for which help will be displayed.
---------	---

**Examples Commands:**

```
help
help rebalance
```

**Sample Output:**

```
gfsh>help rebalance
NAME
    rebalance
IS AVAILABLE
    true
SYNOPSIS
    Rebalance partitioned regions. The default is for all partitioned
region
s to be rebalanced.
SYNTAX
    rebalance [--include-region=value(,value)*]
    [--exclude-region=value(,val
ue)*] [--time-out=value] [--simulate(=value)?]
PARAMETERS
    include-region
        Partitioned regions to be included when rebalancing. Includes
ta
ke precedence over excludes.
        Required:false
    exclude-region
        Partitioned regions to be excluded when rebalancing.
        Required:false
    time-out
        Time to wait (in seconds) before GFSH returns to a prompt
while
rebalancing continues in the background. The default is to wait for
rebalancing
to complete.
        Required:false
        Default if the parameter is not specified:-1
    simulate
        Whether to only simulate rebalancing. The --time-out
parameter i
s not available when simulating.
        Required:false
        Default if no value for the parameter is given:true
        Default if the parameter is not specified:false
```

**hint**

Display information on topics and a list of commands associated with a topic.

Provide hints for a topic or lists all available topics if a topic is not specified.

**Availability:** Online or offline.

**Syntax:**

```
hint [topic]
```

topic	Logical area of GemFire functionality.
-------	--

**Example Commands:**

```
hint
hint Server
```

**Sample Output:**

```
gfsh>hint
Hints are available for the following topics. Use "hint <topic-name>" for
a specific
hint.
Configuration
Data
Debug-Utility
Disk Store
Function Execution
GFSH
Help
JMX
Lifecycle
Locator
Management-Monitoring
Manager
Region
Server
Statistics
WAN

gfsh>hint server
A server is GemFire cluster member which holds a GemFire cache. Depending
on the
topology used it can refer to either a system that responds to client
requests
or a system that is only a peer to other members.

describe member : Display information about a member, including name, id,
groups
, regions, etc.
export logs      : Export the log files for a member or members.
list members     : Display all or a subset of members.
start server     : Start a GemFire Cache Server.
status server    : Display the status of a GemFire Cache Server.
stop server      : Stop a GemFire Cache Server..
```

**history**

Show the command history.

This history can be saved to a file which can also be used as a script later.

A history of commands that have been executed successfully is also logged in `.gfsh.history` file in the home directory of the user running gfsh.

**Availability:** Online or offline.

**Syntax:**

```
history [--file=<history text file>]
```

file	File to which the history is to be saved.
------	---

**Example Commands:**

```
history
history --file=../mycommands.gfsh;
```

**Sample Output:**

```
gfsh>history --file=../mycommands.gfsh
Wrote successfully to file ./mycommands.gfsh
```

**import**

Import data that you have previously exported into a region.

**import data**

Import user data from a file to a region.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
import data --region=value --file=value --member=value
```

region	Required. Region into which data will be imported.
file	Required. File from which the imported data will be read. The file must have an extension of ".gfd" and must be on the file system of the member (or accessible to that member via NFS) that is being targeted for the import.
member	Required. Name/Id of a member which hosts the region. The data will be imported from the specified file on the host where the member is running.

**Example Commands:**

```
import data --region=region2 --file=/mnt5/region2_20121001.gfd
--member=server1
```

**list**

List existing GemFire resources such as deployed applications, disk-stores, functions, members, servers, and regions.

**list deployed**

Display a list of JARs that were deployed to members using [deploy](#) on page 682.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
list deployed [--group=value(,value)*]
```

group	Group(s) of members for which deployed JARs will be displayed. If not specified, JARs for all members will be displayed.
-------	--

**Example Commands:**

```
list deployed
list deployed --group=Group2
```

**Sample Output:**

```
gfsh> list deployed --group=Group2
```

Member	Deployed JAR	JAR Location
-----	-----	-----

```
-----  
datanode1 | group1_functions.jar |  
/usr/local/gemfire/deploy/vf.gf#group1_functions.jar#1  
datanode2 | group1_functions.jar |  
/usr/local/gemfire/deploy/vf.gf#group1_functions.jar#1
```

**Error Messages:****No JAR Files Found****list disk-stores**

List all available disk stores across the GemFire cluster along with their configured disk directories and any Regions, Cache Servers, Gateways, PDX Serialization and Async Event Queues using Disk Stores to either overflow and/or persist information to disk. Use the 'describe disk-store' command to see the details for a particular Disk Store.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:****list disk-stores****Example Commands:****list disk-stores****Sample Output:**

```
gfsh> list disk-stores  
  
Member Name      |           Member Id          | Disk  
Store Name | Disk Store ID  
----- | ----- |  
----- |  
consumerServer | 10.237.177.10(consumerServer:13825)<v5>:3545 |  
consumerData   | 4029af26-fd82-4997-bd6c-33382cdbb5e9 |  
consumerServer | 10.237.177.10(consumerServer:13825)<v5>:3545 |  
observerData   | 7e0316ad-963c-49b0-9b01-8f59b8d9e29e |  
producerServer | 10.237.177.10(producerServer:13826)<v3>:53764 |  
producerData   | 4670e4eb-1c50-4465-b418-08ede3d5dbed |
```

**Error Messages:****gfsh> list disk-stores****No Disk Stores Found****list durable-cqs**

List durable client CQs associated with the specified durable client id.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:****list durable-cqs --durable-client-id=value [--member=value] [--group=value]**

<b>durable-client-id</b>	Required. The ID used to identify the durable client.
<b>member</b>	Name/Id of the member for which the durable client is registered and durable CQs will be displayed.
<b>group</b>	Group of members for which the durable client is registered and durable CQs will be displayed.

**Example Commands:**

```
list durable-cqs --durable-client-id=client1
```

**Sample Output:**

```
gfsh>list durable-cqs --durable-client-id=client1
member | durable-cq-name
----- | -----
server3 | cq3
        | cq1
        | cq2
server4 | cq3
        | cq1
```

**Error Messages:**

```
gfsh>list durable-cqs --durable-client-id=client1
```

Unable to list durable-cqs for durable-client-id : "client1" due to following reasons.

No client found with client-id : client1  
 Occurred on members  
 1.server4  
 2.server1  
 3.server3

**list functions**

Display a list of registered functions. The default is to display functions for all members.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
list functions [--matches=value] [--group=value(,value)*]
[--member=value(,value)*]
```

matches	Pattern that the function ID must match in order to be included. Uses Java pattern matching rules, not UNIX. For example, to match any character any number of times use ".*" instead of "*".
group	Group(s) of members for which functions will be displayed. Use a comma separated list for multiple groups.
member	Name/Id of the member(s) for which functions will be displayed. Use a comma separated list for multiple members.

**Example Commands:**

```
gfsh> list functions
gfsh> list functions --matches=reconcile.*
```

**Sample Output:**

```
list functions
```

Member	Function
camelot	loadDataFromExternalSource
camelot	reconcileWeeklyExpenses
excalibur	loadDataFromExternalSource
excalibur	reconcileDailyExpenses

Example of 'list function' with a "matches" filter:

```
gfsh> list functions --matches=reconcile.*
```

Member	Function
camelot	reconcileWeeklyExpenses
excalibur	reconcileDailyExpenses

Example of 'list functions' when no functions are found in GemFire:

```
gfsh> list functions
No Functions Found.
```

### **list gateways**

Displays the gateway senders and receivers for a member or members.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

#### **Syntax:**

```
list gateways [--group=value(,value)*]
```

group	Group(s) of members for which Gateway Senders and Receivers will be displayed. Use a comma separated list for multiple groups.
-------	--

#### **Example Commands:**

```
list gateways
```

### **list indexes**

Display the list of indexes created for all members.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

#### **Syntax:**

```
list indexes [--with-stats(=value)?]
```

with-stats	Whether statistics should also be displayed. If no value is specified for the parameter, the default value of the parameter is true.
------------	--

#### **Example Commands:**

```
list indexes
```

```
list indexes --with-stats
```

### **Sample Output:**

```
gfsh>list indexes
Member Name | Member ID | Region
-----+-----+-----+-----+-----+-----+
Path | Name | Type | Indexed Expression | From Clause | |
-----+-----+-----+-----+-----+-----+
| ----- | ----- | ----- | ----- | ----- | |
consumerServer | 10.237.177.10(consumerServer:13873):6317 | /consumers
| cidIdx | KEY | id | | /consumers
consumerServer | 10.237.177.10(consumerServer:13873):6317 | /consumers
| cnameIdx | RANGE | name | | /consumers
producerServer | 10.237.177.10(producerServer:13874):19198 | /producers
| pidIdx | RANGE | id | | /producers
```

Example of 'list indexes' with stats printed:

```
gfsh>list indexes --with-stats
```

Member Name Expression	Member ID From Clause	Region Path Uses	Number of Updates	Name Update	Type Time	Indexed Keys	Values
cs...	10...	/consumers	cidx	cidIdx	KEY	id	
/consumers	2512	0	0		5020	5020	
cs...	10...	/consumers	cnameIdx	cnameIdx	RANGE	name	
/consumers	0	5020	421224000		0	5020	
ps...	10...	/producers	pidIdx	pidIdx	RANGE	id	
/producers	0	5031	497872000		5031	5031	

**Error Messages:**

Example of output when no indexes are found in GemFire:

```
gfsh> list indexes
No Indexes Found
```

**list members**

Display all or a subset of members.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
list members [--group=value]
```

group	Group name for which members will be displayed.
-------	---

**Example Commands:**

```
list members
list members --group=Group1
```

**Sample Output:**

```
gfsh>list members
Name | Id
-----
server1 | GemFireStymon(server1:888)<v7>:10839
server2 | GemFireStymon(server2:3260)<v8>:16721
locator1 | localhost(locator1:216:locator):33368
```

**list regions**

Display regions of a member or members. If no parameter is specified, all regions in the GemFire distributed system are listed.

**Syntax:**

```
list regions [--group=value] [--member=value]
```

group	Group of members for which regions will be displayed.
member	Name/Id of the member of the member for which regions will be displayed.

**Example Commands:**

```
list regions
list regions --group=G1
list regions --member=member1
```

**Sample Output:**

```
gfsh>list regions
List of regions
-----
region1
region2
```

**locate**

Locate a region entry on a member.

**locate entry**

Locate a given entry on members using the specified key. This command is useful when using partitioned regions.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
locate entry --key=value --region=value [--key-class=value]
[--value-class=value] [--recursive=value]
```

key	Required. String or JSON text from which to create a key. Examples include: "James", "100L" and "('id': 'l34s')".
region	Required. Region in which to locate values.
key-class	Fully qualified class name of the key's type. The default is java.lang.String.
value-class	Fully qualified class name of the value's type. The default is java.lang.String.
recursive	Whether to traverse regions and subregions recursively. If the value of this parameter is not specified, the default value is false.

**Example Commands:**

```
locate entry --key=('id':'133abg124') --region=/region1
--key-class=data.ProfileKey --recursive=true;
```

**Sample Output:**

```
gfsh>locate entry --key=('123abc') --region=region2
Result          : true
Key Class       : java.lang.String
Key             : ('123abc')
Locations Found : 2

MemberName | MemberId
-----
server1   | GemFireStymon(server1:3692)<v1>:13487
server2   | GemFireStymon(server2:2340)<v2>:11613
```

**netstat**

Report network information and statistics via the "netstat" operating system command.

Report important network usage information/statistics for the given member.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
netstat [--member=value(,value)*] [--group=value] [--file=value]
[--with-lsos(=value)?]
```

member	Name/Id of the member(s) on which to run the netstat command.
group	Group of members on which to run the netstat command.
file	Text file to which output from the netstat command will be written. A ".txt" extension will be added if it's not already a part of the specified name.
with-lsos	Whether lsos (list open files) command output should also be displayed. Not applicable for Microsoft Windows hosts.

**Example Commands:**

```
netstat
netstat --member=server1
netstat --member=server1 --file=server1_netstat.txt
```

**Sample Output:**

```
gfsh>netstat
#####
Host: GemFireStymon
OS: Windows XP 5.1 x86
Member(s):
  server2, locator1, server1
#####
```

## Active Connections

Proto	Local Address	Foreign Address	State
TCP	GemFireStymon:epmap	GemFireStymon:0	LISTENING
TCP	GemFireStymon:microsoft-ds	GemFireStymon:0	LISTENING
TCP	GemFireStymon:1034	GemFireStymon:0	LISTENING
TCP	GemFireStymon:1069	GemFireStymon:0	LISTENING
TCP	GemFireStymon:1099	GemFireStymon:0	LISTENING
TCP	GemFireStymon:1134	GemFireStymon:0	LISTENING
TCP	GemFireStymon:3389	GemFireStymon:0	LISTENING
TCP	GemFireStymon:8080	GemFireStymon:0	LISTENING
TCP	GemFireStymon:8081	GemFireStymon:0	LISTENING
TCP	GemFireStymon:10334	GemFireStymon:0	LISTENING
TCP	GemFireStymon:40404	GemFireStymon:0	LISTENING
TCP	GemFireStymon:40405	GemFireStymon:0	LISTENING
TCP	GemFireStymon:1025	GemFireStymon:0	LISTENING
TCP	GemFireStymon:5152	GemFireStymon:0	LISTENING
TCP	GemFireStymon:netbios-ssn	GemFireStymon:0	LISTENING
TCP	GemFireStymon:1035	GemFireStymon:0	LISTENING
TCP	GemFireStymon:1035	GemFireStymon:1081	ESTABLISHED
TCP	GemFireStymon:1035	GemFireStymon:1086	ESTABLISHED
TCP	GemFireStymon:1035	GemFireStymon:1147	ESTABLISHED
TCP	GemFireStymon:1035	GemFireStymon:1156	ESTABLISHED
TCP	GemFireStymon:1046	GemFireStymon:1099	ESTABLISHED
TCP	GemFireStymon:1049	osdc-proxy-vip.vmware.com:http	CLOSE_WAIT
TCP	GemFireStymon:1050	osdc-proxy-vip.vmware.com:3128	CLOSE_WAIT
TCP	GemFireStymon:1071	GemFireStymon:0	LISTENING
TCP	GemFireStymon:1071	GemFireStymon:1077	ESTABLISHED
TCP	GemFireStymon:1071	GemFireStymon:1150	ESTABLISHED
TCP	GemFireStymon:1071	GemFireStymon:1157	ESTABLISHED
TCP	GemFireStymon:1077	GemFireStymon:1071	ESTABLISHED
TCP	GemFireStymon:1078	GemFireStymon:24400	ESTABLISHED

TCP	GemFireStymon:1081	GemFireStymon:1035	ESTABLISHED
TCP	GemFireStymon:1086	GemFireStymon:1035	ESTABLISHED
TCP	GemFireStymon:1099	GemFireStymon:1046	ESTABLISHED
TCP	GemFireStymon:1136	GemFireStymon:0	LISTENING
TCP	GemFireStymon:1136	GemFireStymon:1143	ESTABLISHED
TCP	GemFireStymon:1136	GemFireStymon:1151	ESTABLISHED
TCP	GemFireStymon:1136	GemFireStymon:1201	ESTABLISHED
TCP	GemFireStymon:1141	GemFireStymon:4247	ESTABLISHED
TCP	GemFireStymon:1142	GemFireStymon:48640	ESTABLISHED
TCP	GemFireStymon:1143	GemFireStymon:1136	ESTABLISHED
TCP	GemFireStymon:1147	GemFireStymon:1035	ESTABLISHED
TCP	GemFireStymon:1150	GemFireStymon:1071	ESTABLISHED
TCP	GemFireStymon:1151	GemFireStymon:1136	ESTABLISHED
TCP	GemFireStymon:1156	GemFireStymon:1035	ESTABLISHED
TCP	GemFireStymon:1157	GemFireStymon:1071	ESTABLISHED
TCP	GemFireStymon:1201	GemFireStymon:1136	ESTABLISHED
TCP	GemFireStymon:1349	GemFireStymon:10334	TIME_WAIT
TCP	GemFireStymon:1350	GemFireStymon:10334	TIME_WAIT
TCP	GemFireStymon:1351	GemFireStymon:10334	TIME_WAIT
TCP	GemFireStymon:1352	GemFireStymon:10334	TIME_WAIT
TCP	GemFireStymon:1353	GemFireStymon:10334	TIME_WAIT
TCP	GemFireStymon:1354	GemFireStymon:10334	TIME_WAIT
TCP	GemFireStymon:4247	GemFireStymon:0	LISTENING
TCP	GemFireStymon:4247	GemFireStymon:1141	ESTABLISHED
TCP	GemFireStymon:24400	GemFireStymon:0	LISTENING
TCP	GemFireStymon:24400	GemFireStymon:1078	ESTABLISHED
TCP	GemFireStymon:48640	GemFireStymon:0	LISTENING
TCP	GemFireStymon:48640	GemFireStymon:1142	ESTABLISHED
UDP	GemFireStymon:microsoft-ds	*:*	
UDP	GemFireStymon:isakmp	*:*	
UDP	GemFireStymon:4500	*:*	
UDP	GemFireStymon:ntp	*:*	
UDP	GemFireStymon:1900	*:*	
UDP	GemFireStymon:ntp	*:*	
UDP	GemFireStymon:netbios-ns	*:*	
UDP	GemFireStymon:netbios-dgm	*:*	
UDP	GemFireStymon:1900	*:*	
UDP	GemFireStymon:32270	*:*	
UDP	GemFireStymon:42838	*:*	
UDP	GemFireStymon:47727	*:*	

**pause gateway-sender**

Pause a gateway sender.

Pause the gateway sender on a member or members.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
pause gateway-sender --id=value [--group=value(,value)*]
[--member=value]
```

id	ID of the gateway sender.
group	Group(s) of members on which to pause the gateway sender.
member	Name/Id of the member on which to pause the gateway sender.

**Example Commands:**

```
pause gateway-sender --id=sender1
```

**put**

Add or update a region entry.

Add or update an entry in a region.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
put --key=value --value=value --region=value [--key-class=value]
[--value-class=value] [--skip-if-exists=value]
```

key	Required. String or JSON text from which to create the key. Examples include: "James", "100L" and "{id: '134s'}".
value	Required. String or JSON text from which to create the value. Examples include: "manager", "100L" and "{value: 'widget'}".
region	Required. Region into which the entry will be put.
key-class	Fully qualified class name of the key's type. The default is java.lang.String.
value-class	Fully qualified class name of the value's type. The default is java.lang.String.
skip-if-exists	Skip the put operation when an entry with the same key already exists. The default is to overwrite the entry (false).

**Example Commands:**

```
put --key=('id':'133abg125')
--value=('firstname':'James','lastname':'Gosling')
--region=/region1 --key-class=data.ProfileKey
--value-class=data.ProfileDetails

put --key=('133abg124') --value=('Hello World!!!') --region=/region2

put --key=('100F') --value=('2146547689879658564')
--region=/region1/region12
--key-class=java.lang.Float --value-class=java.lang.Long
```

**Sample Output:**

```
gfsh>put --key='('123abc')' --value='('Hello World!!!')' --region=region2
Result      : true
Key Class   : java.lang.String
Key         : ('123abc')
Value Class : java.lang.String
Old Value   : <NULL>
```

**Error Messages:**

```
"Region name is either empty or Null";
"Key is either empty or Null";
"Value is either empty or Null";
"Region <{0}> not found in any of the members";
"Region <{0}> Not Found";
"Key is not present in the region";
```

**query**

Run queries against GemFire regions.

Run the specified OQL query as a single quoted string and displays results in pages allows to move between pages. If limit is not set in the query, then a default limit of 1000 (derived from GFSH environment variable

APP\_FETCH\_SIZE) will be applied. Page size is derived from GFSH environment variable APP\_COLLECTION\_LIMIT (default value=20).



**Note:** This command should not be executed from gfsh if the objects being queried contain cyclic references.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

#### Syntax:

```
query --query=value [ --step-name=value ] [ --interactive=value ]
```

query	Required. The OQL string.
step-name	
interactive	Whether or not this query is interactive. If false, then all results will be displayed at once. The default value if this parameter is not specified is true.

#### Example Commands:

```
query --query="select * from /region1/region12" --interactive
\\ Using variables set using 'set variable' for substitution in the query:
query --query="select * from /region1 r1 where r1.id='${PERSON_ID}'"
query --query="select * from ${EMPLOYEE_REGION} el where
el.id='${PERSON_ID}'"
```

#### Sample Output:

```
gfsh>query --query='SELECT * FROM /region2'
Result      : true
startCount  : 0
endCount    : 20
Rows        : 1

Result
-----
('Hello World!!!')

NEXT_STEP_NAME : END
```

## rebalance

Rebalance partitioned regions.

The default is for all partitioned regions to be rebalanced.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

#### Syntax:

```
rebalance [ --include-region=value(,value)* ]
[ --exclude-region=value(,value)* ] [ --time-out=value ] [ --simulate(=value)? ]
```

include-region	Partitioned Region paths to be included for rebalance operation. Includes take precedence over excludes.
exclude-region	Partitioned Region paths to be excluded for rebalance operation.
time-out	Time to wait (in seconds) before GFSH returns to a prompt while rebalancing continues in the background. The default is to wait for rebalancing to complete.
simulate	Whether to only simulate rebalancing. The --time-out parameter is not available when simulating. If no value is specified for the parameter, the default value is true.

**Example Commands:**

```
rebalance --include-region=/region3 --simulate=true
```

**Sample Output:**

```
rebalance
1. server1 host1(3467):12435:12423
Row  Rebalanced Stats          | Value
---  -----
1    TotalBucketCreateBytes    | 0
2    TotalBucketCreateTime    | 0
3    TotalBucketCreatesCompleted | 0
4    TotalBucketTransferBytes | 0
5    TotalBucketTransferTime  | 0
6    TotalBucketTransfersCompleted | 0
7    TotalPrimaryTransferTime | 0
8    TotalPrimaryTransfersCompleted | 0
9    TotalTime                 | 56

Rebalance complete on host1(3467):12435:12423.
```

**remove**

Remove an entry from a region.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
remove --region=value [--key=value] [--all(=value)?] [--key-class=value]
```

key	String or JSON text that will be used to create a key to retrieve a value .
key-class	Fully qualified class name of the key's type. The default is the key constraint for the current region or String.
region	Required. Region from which to remove the entry.
all	Clears the region by removing all entries.

**Example Commands:**

```
gfsh>remove --region=/region1/region12 --key='{"id": "133abg134"}'
gfsh>remove --region=/region1/region12 --key='{"id": "133abg134"}'
--key-class=data.ProfileKey
gfsh>remove --region=/region1/region12 --all
```

**Error Messages:**

```
"Region name is either empty or Null";
"Key is either empty or Null";
"Value is either empty or Null";
"Region <{0}> not found in any of the members";
"Region <{0}> Not Found";
"Key is not present in the region";
"Cleared all keys in the region";
```

**resume gateway-sender**

Resume any gateway senders that you have paused.

Resume the gateway sender on a member or members.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
resume gateway-sender --id=value [--group=value(,value)*]
[--member=value]
```

id	ID of the Gateway Sender.
group	Group(s) of members on which to resume the Gateway Sender.
member	Name/Id of the member on which to resume the Gateway Sender.

**Example Commands:**

```
resume gateway-sender --id=sender1-LN --group=LN-Group1
```

**revoke missing-disk-store**

Instruct the member(s) of a distributed system to stop waiting for a disk store to be available.

Only revoke a disk store if its files are lost as it will no longer be recoverable after revoking is initiated. Use the "show missing-disk-store" command to get descriptions of missing disk stores.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
revoke missing-disk-store --id=value
```

id	Required. ID of the missing disk store to be revoked.
----	---

**Example Commands:**

```
revoke missing-disk-store --id=60399215-532b-406f-b81f-9b5bd8d1b55a
```

**Sample Output:**

```
gfsh>revoke missing-disk-store --id=60399215-532b-406f-b81f-9b5bd8d1b55a
Missing disk store successfully revoked
```

**Error Messages:**

Example of revoke missing-disk-store when the disk store cannot be found:

```
gfsh> revoke missing-disk-store --id=60399215-532b-406f-b81f
```

```
Unable to find missing disk store to revoke
```

**run**

Execute a set of GFSH commands.

Commands that normally prompt for additional input will instead use default values.

**Availability:** Online or offline.



**Note:** Some commands specified in the file require online status.

**Syntax:**

```
run --file=value [--quiet(=value)?] [--continue-on-error(=value)?]
```

file	Required. Path of the file scripted with commands that gfsh recognizes. Path should be relative or absolute.
------	--

quiet	Whether to show command output. If no value is provided for the parameter, the default value is true.   <b>Note:</b> All commands in a script will be run non-interactively with 'run' command. This option doesn't change that.
continue-on-error	Whether further execution of the script should continue if there is an error while executing one of the commands fails. If no value is provided for the parameter, the default value is true.

**Example Commands:**

```
run --file=create-regions.gfsh --quiet=true

(2) From command line:
prompt> /home/user1/gemfire70/bin/gfsh run ./create-regions.gfsh --quiet=true
prompt> /home/user1/gemfire70/bin/gfsh run ./create-regions.gfsh
--continue-on-error=true
```

**Sample Output:**

```
gfsh>run --file=create-regions.gfsh
1. Executing - create region --name=region4 --type=REPLICATE

Member | Status
----- | -----
server2 | Region "/region4" created on "server2"
server1 | Region "/region4" created on "server1"

2. Executing - create region --name=region1/subregion1 --type=LOCAL

Parent region for "region1/subregion1" doesn't exist.
```

**set variable**

Set variables in the GFSH environment.

Set GFSH variables that can be used by commands.

You can use the [echo](#) on page 688 command to view the value of a variable. For example, to see a list of all environment variables and their current values, use the following command:

```
gfsh>echo --string=$*
```

See [Useful gfsh Shell Variables](#) on page 654 for a description of preset environment variables.

**Availability:** Online or offline.

**Syntax:**

```
set variable --name=value --value=value
```

variable	Required. Name for the variable. Name must only be composed of letters, numbers and the "_" character and may not start with a number.
value	Required. Value that the variable will be set to.

**Example Commands:**

```
set variable --name=APP_COLLECTION_LIMIT --value=10
```

```
set variable --name=FOO --value="foo"
set variable --name=BAR --value="bar"
```

**Sample Output:**

```
gfsh>set variable --name=APP_COLLECTION_LIMIT --value=10
Value for variable APP_COLLECTION_LIMIT is now: 10.
```

```
gfsh>set variable --name=BAR --value="bar"
Value for variable BAR is now: "bar".
```

**show**

Display deadlocks, logs, metrics and missing disk-stores.

**show dead-locks**

Display any deadlocks in the GemFire distributed system.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
show dead-locks --file=value
```

file	Required. Name of the file to which dependencies between members will be written.
------	---

**Example Commands:**

```
show dead-locks --file=deadlocks.txt
```

**Sample Output:**

```
gfsh>show dead-locks --file=deadlocks.txt
No dead lock detected.
```

```
Please view the dependencies between the members in file : deadlocks.txt
```

**show log**

Display the log for a member.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
show log --member=value [--lines=value]
```

member	Required. Name/Id of the member whose log file will be displayed.
lines	Number of lines from the log file to display. The maximum is 100. If the parameter is not specified, the default is 30.

**Example Commands:**

```
show log --member=locator1 --lines=5
```

**Sample Output:**

```
gfsh>show log --member=locator1 --lines=5
SystemLog:
[info 2012/09/25 14:04:51.340 PDT locator1 <RMI TCP
Connection(14)-192.168.129.1
29> tid=0x57] (tid=12 msgId=4) Parent region for "region1/subregion1" doesnt
exist.
[info 2012/09/25 14:04:51.372 PDT locator1 <RMI TCP
```

```

Connection(14)-192.168.129.1
29> tid=0x57] (tid=12 msgId=5) Error occurred while executing "create region
--n
ame=region1/subregion1 --type=LOCAL".

[info 2012/09/25 15:14:34.314 PDT locator1 <RMI TCP
Connection(159)-192.168.129.
129> tid=0x68] (tid=13 msgId=6) Error occurred while executing "show log
--membe
r=server1 --lines=5".

```

### show metrics

Display or export metrics for the entire distributed system, a member or a region.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

#### Syntax:

```
show metrics [--member=value] [--region=value] [--file=value]
[--port=value] [--categories=value(,value)*]
```

member	Name/Id of the member whose metrics will be displayed/exported.
region	Name/Path of the region whose metrics will be displayed/exported.
file	Name of the file to which metrics will be written.
port	Port number of the Cache Server whose metrics are to be displayed/exported. This can only be used along with the --member parameter.
categories	Categories available based upon the parameters specified (listed above) are: <ul style="list-style-type: none"> <li>• <b>no parameters specified:</b> cluster, cache, diskstore, query</li> <li>• <b>region specified:</b> cluster, region, partition, diskstore, callback, eviction</li> <li>• <b>member specified:</b> member, jvm, region, serialization, communication, function, transaction, diskstore, lock, eviction, distribution</li> <li>• <b>member and region specified:</b> region, partition, diskstore, callback, eviction</li> </ul>

#### Example Commands:

```

// Metrics for the entire system
show metrics
// Metrics for a region:
show metrics --region=region1
// Metrics for a given member
show metrics --member=server1
// Metrics for a region on a member
show metrics --region=region1 --member=server1
// Metrics for a member and the cacheserver it hosts
// NOTE: port option only work when used with --member option
show metrics --member=server1 --port=10334
// Export metrics for the entire system
show metrics --file=data/stats/system-stats.csv

```

#### Sample Output:

```

gfsh>show metrics

Cluster-wide Metrics

Type      | Metric           | Value
-----|-----|-----
cluster  | totalHeapSize   | 123

```

cache	totalRegionEntryCount	0
	totalRegionCount	0
	totalMissCount	0
	totalHitCount	0
diskstore	totalBytesOnDisk	0
	diskReadsRate	0
	diskWritesRate	0
	flushTimeAvgLatency	0
	totalBackupInProgress	0
query	activeCQCount	0
	queryRequestRate	0

### show missing-disk-stores

Display a summary of the disk stores that are currently missing from a distributed system.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

#### Syntax:

```
show missing-disk-stores
```

#### Example Commands:

```
show missing-disk-stores
```

#### Sample Output:

Disk Store ID	Host	Directory
60399215-532b-406f-b81f-9b5bd8d1b55a	excalibur	/usr/local/gemfire/deploy/disk_store1

### show subscription-queue-size

Shows the number of events in the subscription queue. If a CQ name is provided, it counts the number of events in the subscription queue for the specified CQ.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

#### Syntax:

```
show subscription-queue-size --durable-client-Id=value
[ --durable-cq-name=value ] [ --member=value ] [ --group=value ]
```

durable-client-id	Required. The ID used to identify the durable client.
durable-cq-name	The name that identifies the CQ.
member	Name/Id of the member for which subscription queue events are to be counted.
group	Group of members for which subscription queue events are to be counted.

#### Example Commands:

```
show subscription-queue-size --durable-client-id=client1
```

#### Sample Output:

member	subscription-queue-size for durable-client : "client1".
--------	---

```
server3 | 1  
server4 | 0
```

**Error Messages:**

```
gfsh>show subscription-queue-size --durable-client-Id=client1  
  
No client found with client-id : client1  
Occurred on members  
1.server4  
2.server1  
3.server3
```

**shutdown**

Shut down all members that have a cache.

Asks all the members that have a cache to close the cache and disconnect from the system. The timeout parameter allows you to specify that the system should be shutdown forcibly after the time has exceeded.

This command does not shut down any running locators. Locators must be shut down using the `stop locator` command.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
shutdown [--time-out=value]
```

time-out	Time to wait (in seconds) for a graceful shutdown before members with caches are forcefully stopped.
----------	--

**Example Commands:**

```
shutdown  
shutdown --time-out=15
```

**Sample Output:**

```
gfsh>shutdown  
"As a lot of data in memory will be lost, including possibly events in  
queues, do you really want to shutdown the entire distributed system? (Y/n) :"  
Y  
Successfully shut down
```

**sleep**

Delay gfsh command execution.

Delay for a specified amount of time in seconds - floating point values are allowed.

**Availability:** Online or offline.

**Syntax:**

```
sleep [--time=value]
```

time	Number of Seconds to sleep for. If the time parameter is not specified, a default value of 3 seconds is used.
------	---

**Example Commands:**

```
sleep  
sleep --time=60
```

**Sample Output:**

```
gfsh>sleep --time=60  
gfsh>
```

**start**

Start servers, locators, gateway senders and gateway receivers, and monitoring tools.

**start data-browser**

Launch the DataBrowser tool. See [DataBrowser](#) on page 827 for more information.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
start data-browser
```

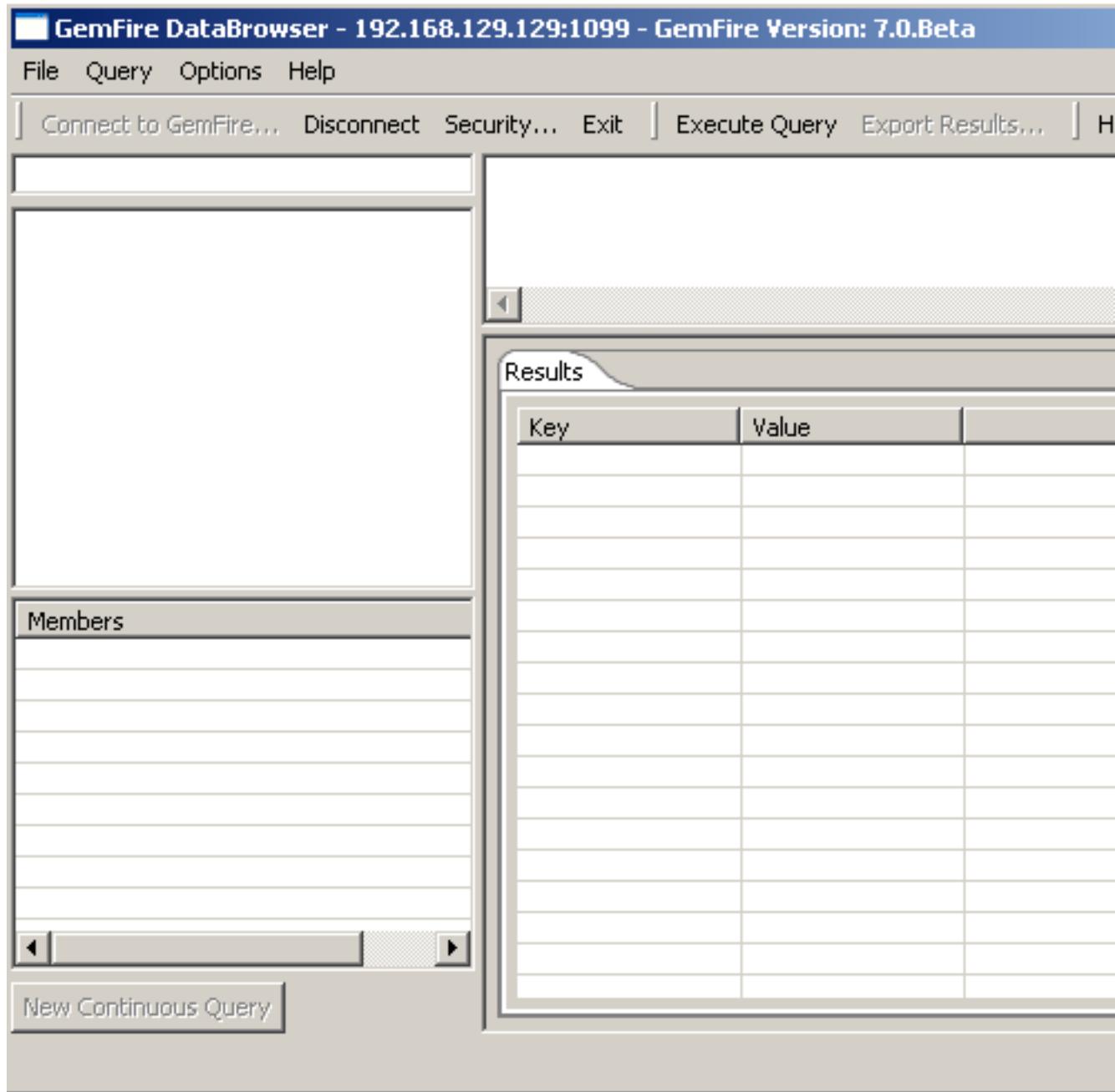
**Example Commands:**

```
start data-browser
```

**Sample Output:**

```
gfsh>start data-browser  
Running GemFire DataBrowser
```

The DataBrowser interface appears with connection information in the status bar:



### **start gateway-receiver**

Start the gateway receiver on a given member or group of members. Note that you can only have one gateway receiver on each member, and unlike a gateway sender, you do not need to specify an identifier for the gateway receiver.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

#### **Syntax:**

```
start gateway-receiver [--group=value(,value)*] [--member=value]
```

member	Name or ID of the member on which to start the Gateway Receiver.
group	Group(s) of members on which to start the Gateway Receiver.

**Example Commands:**

```
start gateway-receiver
start gateway-receiver --member=member1
```

**Sample Output:**

```
gfsh>start gateway-receiver
      Member | Result | Message
-----|-----|
pc13(2266)<v6>:56852 | OK     | GatewayReceiver is started on member
pc13(2266)<v6>:56852
pc13(Manager:2242)<v5>:57631 | Error   | GatewayReceiver is not available on
                                member pc13(Manager:2242)<v5>:57631
pc13(2275)<v7>:47480 | OK     | GatewayReceiver is started on member
pc13(2275)<v7>:47480
pc13(2293)<v8>:55472 | OK     | GatewayReceiver is started on member
pc13(2293)<v8>:55472

gfsh>start gateway-receiver --member=pc13(2266)<v14>:36579
GatewayReceiver is started on member pc13(2266)<v14>:36579

gfsh>start gateway-receiver --group=RG1
      Member | Result | Message
-----|-----|
pc13(2275)<v23>:27484 | OK     | GatewayReceiver is started on member
pc13(2275)<v23>:27484
pc13(2293)<v24>:55810 | OK     | GatewayReceiver is started on member
pc13(2293)<v24>:55810
pc13(2266)<v22>:4522 | OK     | GatewayReceiver is started on member
pc13(2266)<v22>:4522
```

**start gateway-sender**

Start the gateway sender on a member or members. For information on how to configure a gateway sender, see [Configure Gateway Senders](#) on page 196.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
start gateway-sender --id=value [--group=value[,value]*] [--member=value]
```

id	ID of the GatewaySender.
group	Group(s) of members on which to start the Gateway Sender.
member	Member used to start the Gateway Sender

**Example Commands:**

```
start gateway-sender --id=sender1-NY
start gateway-sender --id=sender1-NY --member=server1
start gateway-sender --id=sender1-NY --group=MemberGroup1,MemberGroup2
```

**Sample Output:**

```
gfsh>start gateway-sender --id=ln
      Member | Result | Message
-----|-----|
pc13(30614)<v6>:63670 | OK     | GatewaySender ln is started on
                           member pc13(30614)<v6>:63670
```

```

pc13(30621)<v7>:36015 | OK      | GatewaySender ln is started on
member pc13(30621)<v7>:36015
pc13(30633)<v8>:13633 | OK      | GatewaySender ln is started on
member pc13(30633)<v8>:13633
pc13(Manager:30588)<v5>:42792 | Error   | GatewaySender ln is not available
on member pc13(Manager:30588)<v5>:42792

gfsh>start gateway-sender --id=ln --member=pc13(30614)<v14>:44519
GatewaySender ln is started on member pc13(30614)<v14>:44519
gfsh>start gateway-sender --id=ln --group=SenderGroup1
Member           | Result| Message
----- | -----|
-----
pc13(30614)<v18>:15201 | OK    | GatewaySender ln is started on member
pc13(30614)<v18>:15201
pc13(30621)<v19>:61437 | OK    | GatewaySender ln is started on member
pc13(30621)<v19>:61437
pc13(30633)<v20>:22567 | OK    | GatewaySender ln is started on member
pc13(30633)<v20>:22567

```

### **start jconsole**

Start the JDK JConsole monitoring application in a separate process. JConsole automatically connects to a running JMX Manager node if one is available.

Note that you must have a JDK installed (not just a JRE) and the correct PATH and JAVA\_HOME environment variables set.

See [Browsing GemFire MBeans through JConsole](#) on page 443 for an example of using JConsole with the GemFire management and monitoring system.

**Availability:** Online or offline.

#### **Syntax:**

```
start jconsole [--interval=<seconds>] [--notile] [--version]
[-J<jconsole JVM options>]
```

interval	Set the update interval to n seconds (default is 4 seconds). (Equivalent to JConsole's -interval=n)
notile	Whether to initially tile windows for two or more connections. This parameter is passed as -notile to JConsole.
pluginpath	"Directories or JAR files which are searched for JConsole plugins. The path should contain a provider-configuration file named:\n" + " META-INF/services/com.sun.tools.jconsole.JConsolePlugin\n" + "containing one line for each plugin specifying the fully qualified class name of the class implementing the com.sun.tools.jconsole.JConsolePlugin class."
version	Display the JConsole version information. This parameter is passed as -version to JConsole.
J	Arguments passed to the JVM on which JConsole will run.

#### **Example Commands:**

```

gfsh.> start jconsole --interval=8 --notile;
Running JDK JConsole

gfsh.> start jconsole --version;
JConsole version "1.6.0_31-b04-415"
Java(TM) SE Runtime Environment (build 1.6.0_31-b04-415-11M3646)
Java HotSpot(TM) 64-Bit Server VM (build 20.6-b01-415, mixed mode)

```

#### **Sample Output:**

```

gfsh>start jconsole
Running JDK JConsole

```

The JConsole application appears and auto-connects to a JMX Manager node if one is available:

**Error Messages:**

```
An error occurred while launching JConsole = %1$s  
Connecting by the GemFire member's name or ID is not currently supported.  
Please specify the member as '<hostname|IP>[PORT].  
  
An IO error occurred while launching JConsole.  
Please ensure that JAVA_HOME is set to the JDK installation  
or the JDK bin directory is in the system PATH.  
  
JConsole could not be found.\nPlease ensure that JAVA_HOME is set to the  
JDK installation or the JDK bin directory is in the system PATH.
```

**start jvisualvm**

Start the JDK's Java VisualVM monitoring application in a separate process.

**Availability:** Online or offline.

**Syntax:**

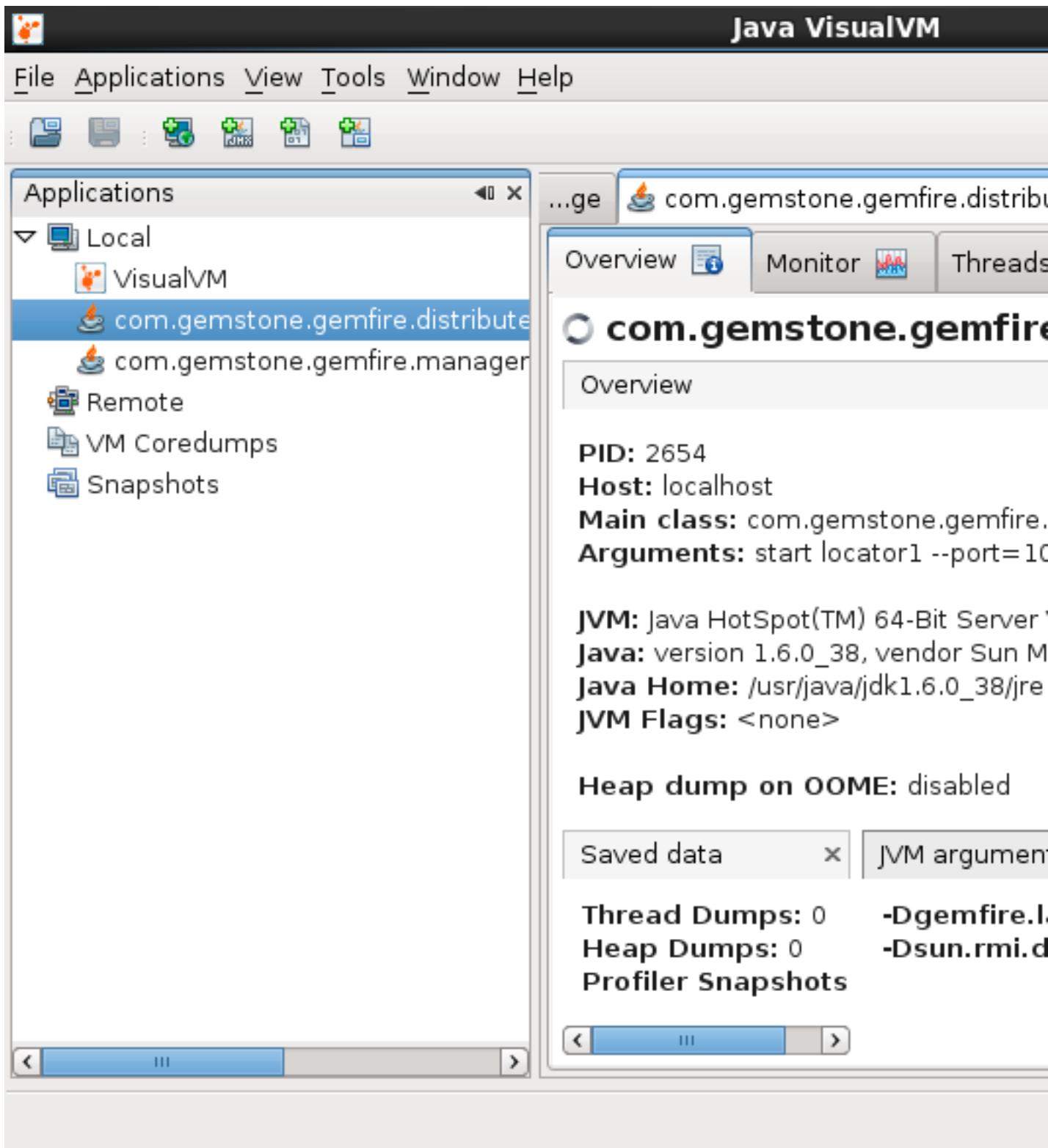
```
start jvisualvm [--J=value(,value)*]
```

J	VM-option passed to the spawned CacheServer VM, example -J-Dfoo.bar=true for setting foo.bar to 'true'.
---	---

**Example Commands:**

```
start jvisualvm
```

**Sample Output:**



## start locator

Start a locator. The command creates a subdirectory and log file named after the locator. If the locator detects that no other JMX Manager exists, then the locator will automatically start an embedded JMX Manager and connect the current gfsh session to the JMX Manager.



**Note:** You must have JAVA\_HOME set before starting gfsh to use this command.

In addition, if gfsh is not already connected to a JMX Manager, the gfsh console will automatically connect to the new embedded JMX Manager started by the new locator.



**Note:** When both --max-heap and --initial-heap are specified during locator startup, additional GC parameters are specified internally by GemFire's Resource Manager. If you do not want the additional default GC properties set by the Resource Manager, then use the -Xms and -Xmx JVM options. See [Control Heap Use with the Resource Manager](#) on page 454 for more information.

**Availability:** Online or offline.

### Syntax:

```
start locator --name=value [--bind-address=value] [--force(=value)?]
[--group=value] [--hostname-for-clients=value] [--locators=value]
[--log-level=value] [--mcast-address=value] [--mcast-port=value]
[--port=value] [--dir=value]
[--properties-file=value] [--security-properties-file=value]
[--initial-heap=value] [--max-heap=value]
[--connect(=value)?] [--J=value(,value)*]
```

name	Required. Name to be used for this GemFire locator service.
bind-address	IP address on which the locator will be bound. (Default: bound to all local addresses)
force	Whether to allow the PID file from a previous locator run to be overwritten.
group	Group(s) the locator will be a part of.
hostname-for-clients	Hostname or IP address that will be sent to clients so they can connect to this locator. (Default: uses the bind-address to which the locator is bound).
locators	List of locators used by this locator to join the appropriate GemFire cluster.
log-level	Level of output logged to the locator log file. Possible values for log-level include: finest, finer, fine, config, info, warning, severe, none.
mcast-address	IP address or hostname used to bind the UPD socket for multi-cast networking so the locator can locate other members in the GemFire cluster. If mcast-port is zero, then mcast-address is ignored.
mcast-port	Port used for multi-cast networking so the locator can locate other members of the GemFire cluster. A zero value disables mcast.
port	Port the locator will listen on. If not specified, the default port is 10334.
dir	Directory in which the Locator will be started and run. The default is ./<locator-member-name>
properties-file	Specify the <code>gemfire.properties</code> file for configuring the locator's distributed system. The file's path should be absolute or relative to gfsh's working directory.
security-properties-file	The <code>gfsecurity.properties</code> file for configuring the Locator's security configuration in the distributed system. The file's path can be absolute or relative to gfsh's working directory.
initial-heap	Size has the same format as the <code>-Xmx/-Xms</code> JVM options.
	<b>Note:</b> If you use the <code>-J-Xms</code> and <code>-J-Xmx</code> JVM properties instead of <code>-initial-heap</code> and <code>-max-heap</code> , then GemFire does not use default JVM resource management properties.

	If you use the JVM properties, you must then specify all properties manually for eviction, garbage collection, heap percentage, and so forth.
max-heap	Size has the same format as the <code>-Xmx/-Xms</code> JVM options   <b>Note:</b> If you use the <code>-J-Xms</code> and <code>-J-Xmx</code> JVM properties instead of <code>-initial-heap</code> and <code>-max-heap</code> , then GemFire does not use default JVM resource management properties. If you use the JVM properties, you must then specify all properties manually for eviction, garbage collection, heap percentage, and so forth.
connect	When connect is set to false, gfsh does not automatically connect to the locator which is started using this command.
J	Argument passed to the JVM on which the Locator will run. For example, specifying <code>--J=-Dfoo.bar=true</code> sets property "foo.bar" to "true".

### Example Commands:

```
start locator --name=locator1
```

### Sample Output:

```
gfsh>start locator --name=locator1
Starting a locator in C:\vFabricGemFire70\Latest\locator1 on
GemFireStymon[10334]
as locator1...
.
.
.
locator in C:\vFabricGemFire70\Latest\locator1 on GemFireStymon[10334] as
locator1
is currently online.
Process ID: 3336
Uptime: 17 seconds
GemFire Version: 7.0
Java Version: 1.6.0_26
Log File: C:\vFabricGemFire70\Latest\locator1\locator1.log
JVM Arguments: -Dgemfire.launcher.registerSignalHandlers=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: ;C:\vFabricGemFire70\Latest\lib\gfsh-dependencies.jar;
C:\vFabricGemFire70\Latest\lib\gemfire.jar;C:\vFabricGemFire70\Latest\lib\antlr.jar;
C:\vFabricGemFire70\Latest\lib\mail.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-core.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-logging-juli.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-jasper.jar;
C:\vFabricGemFire70\Latest\lib\ecj-3.7.2.jar;
C:\vFabricGemFire70\Latest\lib\spring-asm-3.1.1.RELEASE.jar;
C:\vFabricGemFire70\Latest\lib\gemfire.jar;C:\vFabricGemFire70\Latest\lib\antlr.jar;
C:\vFabricGemFire70\Latest\lib\gfsh-dependencies.jar;
C:\vFabricGemFire70\Latest\lib\gfSecurityImpl.jar;
C:\vFabricGemFire70\Latest\SampleCode\tutorial\classes;
C:\vFabricGemFire70\Latest\SampleCode\quickstart\classes;
C:\Program Files\Java\jdk1.6.0_26\lib\tools.jar

Successfully connected to: [host=GemFireStymon, port=1099]
```

### start pulse

Launch the GemFire Pulse monitoring dashboard tool in the user's default system browser and navigates the user to the landing page (login page). For more information on GemFire Pulse, see [GemFire Pulse](#) on page 819.

**Availability:** Online or offline.

**Syntax:**

```
start pulse [--url=value]
```

**Example Commands:**

```
start pulse
start pulse --url=http://gemfire.mycompany.com:8080/pulse
```

**Sample Output:** See [GemFire Pulse](#) on page 819 for examples of Pulse.

**start server**

Start a GemFire cache server process.



**Note:** When both `--max-heap` and `--initial-heap` are specified during locator startup, additional GC parameters are specified internally by GemFire's Resource Manager. If you do not want the additional default GC properties set by the Resource Manager, then use the `-Xms` and `-Xmx` JVM options. See [Control Heap Use with the Resource Manager](#) on page 454 for more information.

**Availability:** Online or offline.

**Syntax:**

```
start server --name=value [--assign-buckets(=value)?]
[--cache-xml-file=value] [--classpath=value]
[--disable-default-server(=value)?]
[--enable-time-statistics(=value)?] [--force(=value)?]
[--properties-file=value] [--security-properties-file=value]
[--group=value][--license-application-cache=value]
[--license-data-management=value]
[--locators=value] [--log-level=value] [--mcast-address=value]
[--mcast-port=value]
[--memcached-port=value] [--memcached-protocol=BINARY|ASCII]
[--rebalance(=value)?]
[--server-bind-address=value] [--server-port=value]
[--statistic-archive-file=value]
[--dir=value] [--initial-heap=value] [--max-heap=value] [--J=value(,value)*]
```

name	Required. Member name for this GemFire Cache Server service.
assign-buckets	Whether to assign buckets to the partitioned regions of the cache on server start.
cache-xml-file	Specifies the name of the XML file or resource to initialize the cache with when it is created.
classpath	Location of user classes required by the Cache Server. This path is appended to the current CLASSPATH.
disable-default-server	Whether the Cache Server will be started by default. If the parameter is specified, the value is set to true. If the parameter is not specified, the value (default) is false.
enable-time-statistics	Causes additional time-based statistics to be gathered for GemFire operations.
properties-file	The <code>gemfire.properties</code> file for configuring the Cache Server's distributed system. The file's path can be absolute or relative to the gfsh working directory.
security-properties-file	The <code>gfsecurity.properties</code> file for configuring the Server's security configuration in the distributed system. The file's path can be absolute or relative to gfsh directory.
group	Group(s) the Cache Server will be a part of.

force	Whether to allow the PID file from a previous Cache Server run to be overwritten.
license-application-cache	Specifies the serial number this distributed system member will use unless a license-data-management serial number is also specified. This distributed member will attempt to activate a GemFire Application Cache Node license which allows it to be a peer-to-peer application cache node. If a license-data-management serial number is provided, then the license-application-cache serial number may be specified as the license for cache clients connecting to this node.
license-data-management	Specifies the serial number(s) this distributed system member will use. This distributed member will attempt to activate a GemFire Data Management Node license which allows it to provide advanced data management services which includes hosting a cache server to which cache clients may connect.
locators	Sets the list of locators used by the Cache Server to join the appropriate GemFire cluster.
log-level	Sets the level of output logged to the Cache Server log file. Possible values for log-level include: finest, finer, fine, config, info, warning, severe, none.
mcast-address	The IP address or hostname used to bind the UDP socket for multi-cast networking so the Cache Server can locate other members in the GemFire cluster. If mcast-port is zero, then mcast-address is ignored.
mcast-port	Sets the port used for multi-cast networking so the Cache Server can locate other members of the GemFire cluster. A zero value disables mcast.
memcached-port	If specified and is non-zero, sets the port number for an embedded Gemcached server and starts the Gemcached server.
memcached-protocol	Sets the protocol used by an embedded Gemcached server. Valid values are BINARY and ASCII . If you omit this property, the ASCII protocol is used.
server-bind-address	IP address on which the Server will be bound. If not specified, the default is to bind to all local addresses.
server-port	Port the Server will listen on for client connections. If the option is not specified, the default is port 40404.
rebalance	Whether to initiate rebalancing across the GemFire cluster. If the parameter is specified without a value, the value is set to true. If the parameter is not specified, the value (default) is false.
dir	Specify the directory in which the server will run in. The default (if not specified) is a directory named after the server. This directory is written to the location where you started gfsh.
statistic-archive-file	The file that statistic samples are written to. An empty string (default) disables statistic archival.
initial-heap	Initial size of the heap in the same format as the JVM -Xms parameter.   <b>Note:</b> If you use the <code>-J-Xms</code> and <code>-J-Xmx</code> JVM properties instead of <code>-initial-heap</code> and <code>-max-heap</code> , then GemFire does not use default JVM resource management properties. If you use the JVM properties, you must then specify all properties manually for eviction, garbage collection, heap percentage, and so forth.

max-heap	Maximum size of the heap in the same format as the JVM -Xmx parameter.
J	Note: If you use the <code>-J-Xms</code> and <code>-J-Xmx</code> JVM properties instead of <code>-initial-heap</code> and <code>-max-heap</code> , then GemFire does not use default JVM resource management properties. If you use the JVM properties, you must then specify all properties manually for eviction, garbage collection, heap percentage, and so forth.

## Examples

```
gfsh>start server --name=server1
gfsh>start server --name=server2 --server-port=40405
```

### Sample Output:

```
gfsh>start server --name=server1
Starting a Cache server in C:\vFabricGemFire70\Latest\server1 on
GemFireStymon[40404]
as server1...
.....
Server in C:\vFabricGemFire70\Latest\server1 on GemFireStymon[40404] as
server1
is currently online.
Process ID: 3828
Uptime: 8 seconds
GemFire Version: 7.0
Java Version: 1.6.0_26
Log File: C:\vFabricGemFire70\Latest\server1\server1.log
JVM Arguments: -Dgemfire.launcher.registerSignalHandlers=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: ;C:\vFabricGemFire70\Latest\lib\gfsh-dependencies.jar;
C:\vFabricGemFire70\Latest\lib\gemfire.jar;C:\vFabricGemFire70\Latest\lib\antlr.jar;
C:\vFabricGemFire70\Latest\lib\mail.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-core.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-logging-juli.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-jasper.jar;
C:\vFabricGemFire70\Latest\lib\ecj-3.7.2.jar;
C:\vFabricGemFire70\Latest\lib\spring-asm-3.1.1.RELEASE.jar;
C:\vFabricGemFire70\Latest\lib\gemfire.jar;C:\vFabricGemFire70\Latest\lib\antlr.jar;
C:\vFabricGemFire70\Latest\lib\gfsh-dependencies.jar;
C:\vFabricGemFire70\Latest\lib\gfSecurityImpl.jar;
C:\vFabricGemFire70\Latest\SampleCode\tutorial\classes;
C:\vFabricGemFire70\Latest\SampleCode\quickstart\classes;
C:\Program Files\Java\jdk1.6.0_26\lib\tools.jar
```

### start vsd

Launch GemFire Visual Statistics Display (VSD) in a separate process. You can specify a comma delimited list of directories and specific GemFire statistics archive files (.gfs) to load into VSD upon start. A directory locations will be searched recursively for statistics archive (.gfs) files.

**Availability:** Online or offline.

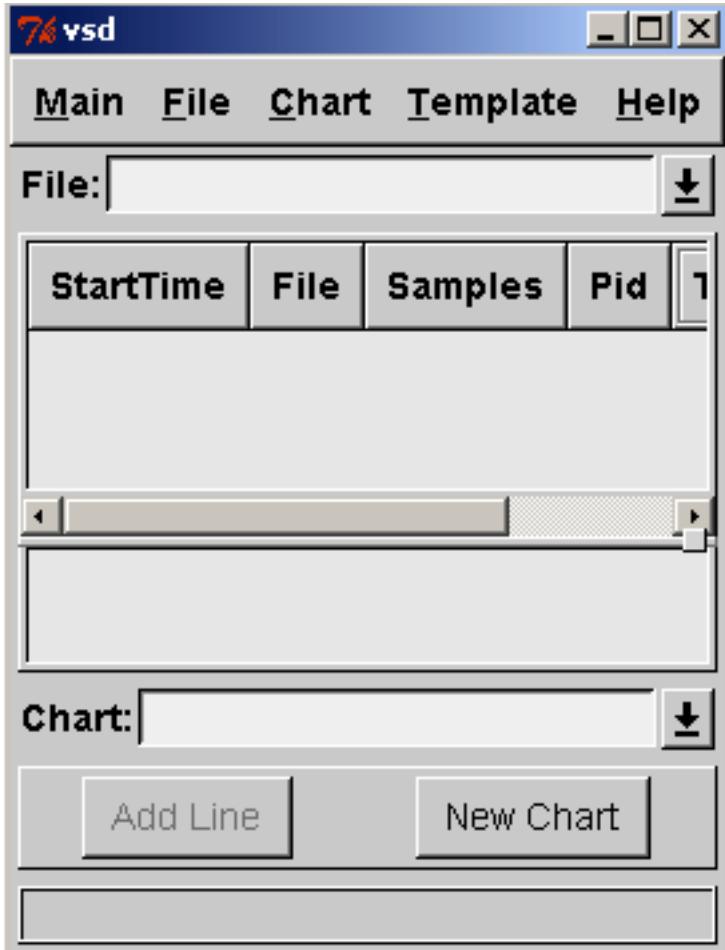
**Syntax:**

```
start vsd [/path/to/dir/containing/gfs/files|/path/to/statisticsArchive.gfs]
[./path/to/dir/containing/gfs/files|/path/to/statisticsArchive.gfs]*
```

**Example Commands:**

```
gfsh> start vsd
/export/gemfire/node1,/export/gemfire/node2/statArchive.gfs,.../gemfire/nodeN;
Running GemFire Visual Statistics Display (VSD)

gfsh> start vsd;
Running GemFire Visual Statistics Display (VSD)
```

**Sample Output:****status**

Check the status of GemFire member processes, including gateway receivers, gateway senders, locators, and servers.

**status gateway-receiver**

Display the status of the specified gateway receiver.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
status gateway-receiver [--group=value(,value)*] [--member=value]
```

group	Group(s) of Gateway Receivers for which to display status.
member	Name/Id of the Gateway Receiver for which to display status.

**Example Commands:**

```
status gateway-receiver --group=LN-Group1
status gateway-receiver --member=server1
```

**Sample Output:**

```
gfsh>status gateway-receiver
Member | Port | Status
-----|-----|-----
pc13(8151)<v2>:26518 | 26837 | Running
pc13(8175)<v4>:53787 | 23753 | Running
pc13(8164)<v3>:24081 | 25457 | Running

Member | Error
-----|-----
pc13(Manager:8124)<v1>:52410 | GatewayReceiver is not available or already stopped
pc13(8130):8180 | GatewayReceiver is not available or already stopped

gfsh>status gateway-receiver --member=pc13(8151)<v2>:50130
Member | Port | Status
-----|-----|-----
pc13(8151)<v2>:50130 | 28592 | Running

gfsh>status gateway-receiver --group=RG1
Member | Port | Status
-----|-----|-----
pc13(8151)<v2>:19450 | 27815 | Running
pc13(8175)<v4>:14139 | 27066 | Running
pc13(8164)<v3>:45150 | 29897 | Running
```

**status gateway-sender**

Display the status of the specified gateway sender.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
status gateway-sender --id=value [--group=value(,value)*]
[--member=value]
```

id	Required. ID of the Gateway Sender.
group	Group(s) of Gateway Senders for which to display status. Comma separated list for multiple member groups.
member	Name/Id of the Gateway Sender for which to display status.

**Example Commands:**

```
status gateway-sender receiver1-LN --group=LN-Group1;
```

**Sample Output:**

```
gfsh>status gateway-sender --id=ln_Serial
Member          | Type   | Runtime Policy | Status
-----|-----|-----|-----
pc13(8175)<v4>:21449 | Serial | Secondary    | Not Running
pc13(8151)<v2>:40761 | Serial | Secondary    | Not Running
pc13(8164)<v3>:33476 | Serial | Secondary    | Not Running

Member          | Error
-----|-----
pc13(8130):2365 | GatewaySender is not available
pc13(Manager:8124)<v1>:43821 | GatewaySender is not available

gfsh>status gateway-sender --id=ln_Serial --member=pc13(8151)<v2>:7411
Member          | Type   | Runtime Policy | Status
-----|-----|-----|-----
pc13(8151)<v2>:7411 | Serial | Secondary    | Not Running

gfsh>status gateway-sender --id=ln_Serial --member=pc13(8151)<v2>:7411
Member          | Type   | Runtime Policy | Status
-----|-----|-----|-----
pc13(8151)<v2>:7411 | Serial | Primary      | Running

gfsh>status gateway-sender --id=ln_Serial --group=Serial_Sender
==>
Member          | Type   | Runtime Policy | Status
-----|-----|-----|-----
pc13(8151)<v2>:44396 | Serial | Secondary    | Not Running
pc13(8164)<v3>:29475 | Serial | Secondary    | Not Running
Member          | Error
-----|-----
pc13(8186)<v5>:45840 | GatewaySender is not available
```

**status locator**

Displays the status of the specified locator. The status will be one of the following:

- started
- online
- offline
- not responding

**Availability:** Online or offline.

**Syntax:**

```
status locator [--name=value] [--host=value] [--port=value]
[--pid=value] [--dir=value]
```

name	Name/Id of the locator for which to display status.
host	Hostname or IP address on which the Locator is running.
port	Port on which the locator is listening. The default is 10334.
pid	Process ID (PID) of the running locator.
dir	Directory in which the locator was started. The default is the current directory.

**Example Commands:**

```
status locator
status locator --name=locator1
```

**Sample Output:**

```
gfsh>status locator --dir=locator1
Locator in C:\vFabricGemFire70\Latest\locator1 on GemFireStymon[10334] as
locator1
is currently online.
Process ID: 3336
Uptime: 2 hours 15 seconds
GemFire Version: 7.0
Java Version: 1.6.0_26
Log File: C:\vFabricGemFire70\Latest\locator1\locator1.log
JVM Arguments: -Dgemfire.launcher.registerSignalHandlers=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: ;C:\vFabricGemFire70\Latest\lib\gfsh-dependencies.jar;
C:\vFabricGemFire70\Latest\lib\gemfire.jar;C:\vFabricGemFire70\Latest\lib\antlr.jar;
C:\vFabricGemFire70\Latest\lib\mail.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-core.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-logging-juli.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-jasper.jar;
C:\vFabricGemFire70\Latest\lib\ecj-3.7.2.jar;
C:\vFabricGemFire70\Latest\lib\spring-asm-3.1.1.RELEASE.jar;
C:\vFabricGemFire70\Latest\lib\gemfire.jar;C:\vFabricGemFire70\Latest\lib\antlr.jar;
C:\vFabricGemFire70\Latest\lib\gfsh-dependencies.jar;
C:\vFabricGemFire70\Latest\lib\gfSecurityImpl.jar;
C:\vFabricGemFire70\Latest\SampleCode\tutorial\classes;
C:\vFabricGemFire70\Latest\SampleCode\quickstart\classes;
C:\Program Files\Java\jdk1.6.0_26\lib\tools.jar
```

**status server**

Display the status of the specified GemFire cache server.

**Availability:** Online or offline.

**Syntax:**

```
status server [--name=value] [--pid=value] [--dir=value]
```

name	Name/Id of the Cache Server for which to display status.
pid	Process ID (PID) of the running GemFire Cache Server.
dir	Directory in which the GemFire Cache Server was started. The default is the current directory.

**Example Commands:**

```
status server
status server --name=server1
```

**Sample Output:**

```
gfsh>status server --name=server1
Server in C:\vFabricGemFire70\Latest\server1 on GemFireStymon[40404] as
server1
is currently online.
Process ID: 3828
Uptime: 3 hours 23 seconds
GemFire Version: 7.0
Java Version: 1.6.0_26
Log File: C:\vFabricGemFire70\Latest\server1\server1.log
JVM Arguments: -Dgemfire.launcher.registerSignalHandlers=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: ;C:\vFabricGemFire70\Latest\lib\gfsh-dependencies.jar;
C:\vFabricGemFire70\Latest\lib\gemfire.jar;C:\vFabricGemFire70\Latest\lib\antlr.jar;
C:\vFabricGemFire70\Latest\lib\mail.jar;
```

```
C:\vFabricGemFire70\Latest\lib\tomcat-embed-core.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-logging-juli.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-jasper.jar;
C:\vFabricGemFire70\Latest\lib\ecj-3.7.2.jar;
C:\vFabricGemFire70\Latest\lib\spring-asm-3.1.1.RELEASE.jar;
C:\vFabricGemFire70\Latest\lib\gemfire.jar;C:\vFabricGemFire70\Latest\lib\antlr.jar;
C:\vFabricGemFire70\Latest\lib\gfsh-dependencies.jar;
C:\vFabricGemFire70\Latest\lib\gfSecurityImpl.jar;
C:\vFabricGemFire70\Latest\SampleCode\tutorial\classes;
C:\vFabricGemFire70\Latest\SampleCode\quickstart\classes;
C:\Program Files\Java\jdk1.6.0_26\lib\tools.jar
```

**stop**

Stop gateway receivers, gateway senders, locators, and servers.

**stop gateway-receiver**

Stop the gateway receiver on a member or members.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
stop gateway-receiver [--group=value(,value)*] [--member=value]
```

group	Group(s) of members on which to stop the Gateway Receiver. Use a comma-separated list for multiple member groups.
member	Name/Id of the member on which to stop the Gateway Receiver.

**Example Commands:**

```
stop gateway-receiver --member=receiver1-LN
stop gateway-receiver --group=LN-Group1
```

**Sample Output:**

```
gfsh>stop gateway-receiver
      Member          | Result | Message
-----|-----|
pc13(2266)<v6>:56852 | OK     | GatewayReceiver is stopped on member
pc13(2266)<v6>:56852
pc13(Manager:2242)<v5>:57631 | Error   | GatewayReceiver is not available on
                                member pc13(Manager:2242)<v5>:57631
pc13(2275)<v7>:47480    | OK     | GatewayReceiver is stopped on member
pc13(2275)<v7>:47480
pc13(2293)<v8>:55472    | OK     | GatewayReceiver is stopped on member
pc13(2293)<v8>:55472

gfsh>stop gateway-receiver --member=pc13(2266)<v14>:36579
GatewayReceiver is stopped on member pc13(2266)<v14>:36579

gfsh>stop gateway-receiver --group=RG1
      Member          | Result | Message
-----|-----|
pc13(2275)<v23>:27484 | OK     | GatewayReceiver is stopped on member
pc13(2275)<v23>:27484
pc13(2293)<v24>:55810 | OK     | GatewayReceiver is stopped on member
pc13(2293)<v24>:55810
```

```
pc13(2266)<v22>:4522 | OK      | GatewayReceiver is stopped on member
pc13(2266)<v22>:4522
```

### **stop gateway-sender**

Stop a gateway sender with a given id on a specified member or members of a specified member group.

**Availability:** Online. You must be connected in gfsh to a JMX Manager member to use this command.

**Syntax:**

```
stop gateway-sender --id=value [--group=value(,value)*] [--member=value]
```

id	Required. ID of the Gateway Sender.
group	Group(s) of members on which to stop the Gateway Sender.
member	Name/Id of the member on which to stop the Gateway Sender.

**Example Commands:**

```
stop gateway-sender --id=ln --member=server1
```

**Sample Output:**

```
gfsh>stop gateway-sender --id=ln
Member          | Result | Message
-----|-----|-----
pc13(5184)<v7>:41914 | OK    | GatewaySender ln is stopped on
member pc13(5184)<v7>:41914
pc13(5192)<v8>:25704 | OK    | GatewaySender ln is stopped on
member pc13(5192)<v8>:25704
pc13(5174)<v6>:53996 | OK    | GatewaySender ln is stopped on
member pc13(5174)<v6>:53996
pc13(Manager:5148)<v5>:64040 | Error | GatewaySender ln is not available
on member pc13(Manager:5148)<v5>:64040
gfsh>stop gateway-sender --id=ln --member=pc13(5174)<v14>:17819
GatewaySender ln is stopped on member pc13(5174)<v14>:17819

gfsh>stop gateway-sender --id=ln --group=SenderGroup1
Member          | Result | Message
-----|-----|-----
pc13(5174)<v18>:63332 | OK    | GatewaySender ln is stopped on member
pc13(5174)<v18>:63332
pc13(5184)<v19>:20055 | OK    | GatewaySender ln is stopped on member
pc13(5184)<v19>:20055
pc13(5192)<v20>:14622 | OK    | GatewaySender ln is stopped on member
pc13(5192)<v20>:14622
```

### **stop locator**

Stop a locator.

**Availability:** Online or offline. If you want to stop a locator while you are offline, use the --dir parameter.

**Syntax:**

```
stop locator [--name=value] [--host=value] [--port=value] [--pid=value]
[--dir=value]
```

name	The GemFire member name or id of the Locator to stop.
------	---

pid	The process id (PID) of the running Locator.
dir	Directory in which the Locator was started. The default is the current directory.

**Example Commands:**

```
stop locator
stop locator --name=locator1
stop locator --dir=locator1
stop locator --pid=6716
```

**Sample Output:**

```
gfsh>stop locator --pid=6716
Locator in c:\vFabricGemFire70\Latest on localhost/127.0.0.1[10334] has
been requested to stop.

//If you stop a locator that you're currently connected to, you will receive
a disconnect message:
gfsh>
No longer connected to GemFireStymon[1099].
```

**stop server**

Stop a GemFire cache server.

**Availability:** Online or offline. If you want to stop a cache server while you are offline, use the `--dir` parameter.

**Syntax:**

```
stop server [ --name=value ] [ --pid=value ] [ --dir=value ]
```

name	Name/Id of the GemFire Cache Server to stop.
pid	Process ID (PID) of the running GemFire Cache Server.
dir	Directory in which the GemFire Cache Server was started. The default is the current directory.

**Example Commands:**

```
stop server --name=server1
stop server --dir=server1
stop server --pid=8540
```

**Sample Output:**

```
gfsh>stop server --pid=8540
Server in c:\vFabricGemFire70\Latest on localhost/127.0.0.1[40404] has been
requested to stop.
```

**undeploy**

Undeploy the JAR files that were deployed on members or groups using `deploy` command.

If `--jar` is not specified, the command will undeploy all deployed JARs. If `--group` is not specified, the command applies to the entire cluster. Note that this command can't unload the classes that were loaded during deployment. Member(s) should be restarted for that.

**Availability:** Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

**Syntax:**

```
undeploy [ --jar=<jar to be undeployed> ] [ --group=<member-group-name> ]
```

group	Group(s) from which the specified JAR(s) will be undeployed. If not specified, undeploy will occur on all members.
jar	JAR or JARs to be undeployed. If not specified, all JARs will be undeployed.

**Example Commands:**

```
undeploy --jar=domain-objects.jar
undeploy --group=Group1
```

**Sample Output:**

```
gfsh>undeploy --jar=domain-objects.jar

Member | Un-Deployed JAR | Un-Deployed From JAR Location
----- | ----- | -----
datanode10 | domain-objects.jar | /usr/local/gemfire/deploy/GF#domain-objects#1
datanode11 | domain-objects.jar | /usr/local/gemfire/deploy/GF#domain-objects#1

gfsh> undeploy --group=Group1

Member | Un-Deployed JAR | Un-Deployed From JAR
Location
----- | ----- | -----
datanode1 | group1_functions.jar | /usr/local/gemfire/deploy/GF#group1_functions.jar#1
datanode1 | group1_dependencies.jar | /usr/local/gemfire/deploy/GF#group1_dependencies.jar#1
datanode2 | group1_functions.jar | /usr/local/gemfire/deploy/GF#group1_functions.jar#1
datanode2 | group1_dependencies.jar | /usr/local/gemfire/deploy/GF#group1_dependencies.jar#1
```

**Error Messages:**

```
No JAR Files Found
```

**upgrade offline-disk-store**

Upgrade offline disk-stores used in vFabric GemFire 6.5 or 6.6 installations to a format that is compatible with vFabric GemFire 7.0.

If the disk store is large, additional memory may need to be allocated to the process using the `--J=-Xmx???` parameter.

**Availability:** Offline.

**Syntax:**

```
upgrade offline-disk-store --name=value --disk-dirs=value(,value)*
[ --max-oplog-size=value ] [ --J=value(,value)* ]
```

name	Required. Name of the offline disk store to be upgraded.
disk-dirs	Required. Directories where data for the disk store was previously written.
max-oplog-size	Maximum size (in megabytes) of the oplogs created by the upgrade. The default value if the parameter is not specified is -1.

J

Arguments passed to the Java Virtual Machine performing the upgrade operation on the disk store.

**Example Commands:**

```
upgrade offline-disk-store --name=DiskStore1 --disk-dirs=/Disks/DiskStore1
upgrade offline-disk-store --name=DiskStore1 --disk-dirs=/Disks/DiskStore1
--J=-DXmx512
```

**validate offline-disk-store**

Validate offline disk stores.

**Availability:** Offline.**Syntax:**

```
validate offline-disk-store --name=value --disk-dirs=value(,value)*
```

name	Required. Name of the disk store to be validated.
disk-dirs	Required. Directories where data for the disk store was previously written.

**Example Commands:**

```
validate offline-disk-store --name=DiskStore2 --disk-dirs=data/dir3,data/dir4
```

**version**

Display product version information.

**Availability:** Online or offline.**Syntax:**

```
version [-full]
```

full	Whether to show the full version information.
------	---

**Example Commands:**

```
version
version --full
```

**Sample Output:**

```
gfsh>version
v7.0Beta
gfsh>version --full
Java version: 7.0 build 38623 10/16/2012 09:37:20 PDT javac 1.6.0_26
Native version: 6.5 06/02/2010 11:23:23 PDT optimized x86 Windows XP 5.1
Source revision: 38623
Source repository: gemfire/trunk
Running on: GemFireStymon/192.168.129.129, 1 cpu(s), x86 Windows XP 5.1
```

## Creating and Running gfsh Command Scripts

gfsh offers several ways to run commands in a scripting environment.

### Running gfsh Scripts

You can create and run scripts that contain gfsh commands that you wish to execute. To execute the script, use the gfsh [run](#) on page 707 command. For example:

```
gfsh run --file=mycommands.gfsh
```



**Note:** When you run a gfsh script, interactive parameters are ignored. You can also set the script to run in quiet mode to prevent output and instruct the script to skip any errors it encounters.

Your command history file can be helpful when you write a gfsh script. A history of commands that have been executed successfully is logged in the `.gfsh.history` file in the home directory of the user running gfsh. You can also export a history file with the `history --file=your_file_name` command.

### Running gfsh Commands on the OS Command Line

You can run some gfsh commands directly from your operating system's prompt by preceding the command with `gfsh`. This can be useful for Unix shell or Windows batch scripting. For example:

```
C:\vFabricGemFire70\Latest>gfsh start locator --name=locator2 --port=10335
```

If the command is successful, the output appears and the operating system prompt reappears:

```
.....
Locator in C:\vFabricGemFire70\Latest\locator2 on GemFireStymon[10335] as
locator2 is currently online.
Process ID: 4000
Uptime: 12 seconds
GemFire Version: 7.0
Java Version: 1.6.0_26
Log File: C:\vFabricGemFire70\Latest\locator2\locator2.log
JVM Arguments: -Dgemfire.launcher.registerSignalHandlers=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: ;C:\vFabricGemFire70\Latest\lib\gfsh-dependencies.jar;
C:\vFabricGemFire70\Latest\lib\gemfire.jar;C:\vFabricGemFire70\Latest\lib\antlr.jar;
C:\vFabricGemFire70\Latest\lib\mail.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-core.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-logging-juli.jar;
C:\vFabricGemFire70\Latest\lib\tomcat-embed-jasper.jar;
C:\vFabricGemFire70\Latest\lib\ecj-3.7.2.jar;
C:\vFabricGemFire70\Latest\lib\spring-asm-3.1.1.RELEASE.jar;
C:\vFabricGemFire70\Latest\lib\gemfire.jar;C:\vFabricGemFire70\Latest\lib\antlr.jar;
C:\vFabricGemFire70\Latest\lib\gfsh-dependencies.jar;
C:\vFabricGemFire70\Latest\lib\gfSecurityImpl.jar;
C:\vFabricGemFire70\Latest\SampleCode\tutorial\classes;
C:\vFabricGemFire70\Latest\SampleCode\quickstart\classes;
C:\Program Files\Java\jdk1.6.0_26\lib\tools.jar
prompt>
```

To see which gfsh commands are available directly on the prompt:

```
prompt# gfsh help
```

The following output appears:

```
vFabric GemFire(TM) v7.0.1 Command Line Shell

USAGE
gfsh [ <command [options]> | <help [command] | [-e "<command> [options]" ]* ]
]

AVAILABLE COMMANDS
compact offline-disk-store
    Compact an offline disk store. If the disk store is large, additional
memory
    may need to be allocated to the process using the --J=-Xmx??? parameter.
describe offline-disk-store
```

```

    Display information about an offline disk store.
encrypt password
    Encrypt a password for use in data source configuration.
run
    Execute a set of GFSH commands. Commands that normally prompt for
additional
    input will instead use default values.
start jconsole
    Start the JDK's JConsole tool in a separate process. JConsole will be
launch
ed, but connecting
    to GemFire must be done manually.
start jvisualvm
    Start the JDK's Java VisualVM (jvisualvm) tool in a separate process.
Java
VisualVM will be launched, but connecting to GemFire must be done manually.
start locator
    Start a Locator.
start pulse
    Open a new window in the default Web browser with the URL for the Pulse
appl
ication.
start server
    Start a GemFire Cache Server.
start vsd
    Start VSD in a separate process.
status locator
    Display the status of a Locator. Possible statuses are: started, online,
off
line or not
    responding.
status server
    Display the status of a GemFire Cache Server.
stop locator
    Stop a Locator.
stop server
    Stop a GemFire Cache Server.
validate offline-disk-store
    Scan the contents of a disk store to verify that it has no errors.
version
    Display product version information.

EXAMPLES
gfsh
    Start GFSH in interactive mode.
gfsh help start locator
    Display help for the "start locator" command.
gfsh start locator --name=locator1
    Start a Locator with the name "locator1".
gfsh -e "connect" -e "list members"
    Connect to a running Locator using the default connection information
and
run the "list
members" command.

```

To summarize, you can run the following gfsh commands directly from the operating system shell:

- gfsh compact offline-disk-store
- gfsh describe offline-disk-store
- gfsh encrypt password

- gfsh run
- gfsh start jconsole
- gfsh start jvisualvm
- gfsh start locator
- gfsh start pulse
- gfsh start server
- gfsh start vsd
- gfsh status locator
- gfsh status server
- gfsh stop locator
- gfsh stop server
- gfsh validate offline-disk-store
- gfsh version

To run multiple commands directly on the command line:

```
prompt>gfsh -e "start locator --name=locator1" -e "start server  
--name=server1"  
  
prompt>gfsh -e "start jconsole"  
  
prompt>gfsh -e "connect --locators=remotehost[10334]" -e "rebalance" -e  
"gc"
```

# Chapter 52

---

## vFabric GemFire Locator Process

The locator is a vFabric GemFire process that tells new, connecting members where running members are located and provides load balancing for server use.

You can run locators as peer locators, server locators, or both:

- Peer locators give joining members connection information to members already running in the locator's distributed system.
- Server locators give clients connection information to servers running in the locator's distributed system. Server locators also monitor server load and send clients to least-loaded servers.

By default, locators run as peer and server locators.

You can run the locator standalone or co-located inside another GemFire process. Running your locators standalone provides the highest reliability and availability of the locator service as a whole.

You must have tools.jar (included with the JDK) added to your CLASSPATH in order to manage servers using either gfsh or the LocatorLauncher API.

### Locator Configuration and Log Files

Locator configuration and log files have the following properties:

- Locators are members of the distributed system just like any other member. In terms of mcast-port and locators configuration, a locator should be configured in the same manner as a server. Therefore, if there are two locators in the distributed system, each locator should reference both locators (just like a server member would).
- You can configure locators by using gemfire.properties or by specifying start-up parameters on the command line. If you are specifying the locator's configuration in a properties file, locators require the same gemfire.properties settings as other members of the distributed system and the same gfsecurity.properties settings if you are using a separate, restricted access security settings file.

For example, to configure both locators and a multicast port in gemfire.properties :

```
locators=host1[port1],host2[port2]
mcast-port=0
```

The same configuration on the gfsh command line:

```
gfsh>start locator --name=locator1
```

- There is no cache configuration specific to locators.
- For logging output, the locator creates a log file in its current working directory. Log file output defaults to locator\_name.log in the locator's working directory. If you restart a locator with a previously used locator name, the existing locator\_name.log file is automatically renamed for you (for example, locator1-01-01.log or locator1-02-01.log). You can modify the level of logging details in this file by specifying a level in the --log-level argument when starting up the locator.

- By default, the Locator will start in a subdirectory (named after the Locator) under the directory where gfsh is executed. This subdirectory is considered the current working directory. You can also specify a different working directory when starting the Locator in gfsh.

## Start the Locator

Use the following steps to guidelines to start the locator:

- **Standalone locator.** Start a standalone locator in one of these ways:
  - Use the gfsh command-line utility. See [Using the GemFire SHell \(gfsh\)](#) on page 641 for more information on using gfsh. The syntax for starting the

Example commands:

```
gfsh>start locator --name=locator1  
gfsh> start locator --name=locator2 --bind-address=192.168.129.205 --port=13489
```

- Start the locator using the main method in the com.gemstone.gemfire.distributed.LocatorLauncher class and the Java executable. For example:

```
java -cp  
<path-to-gemfire.jar-and-application-classes>  
com.gemstone.gemfire.distributed.LocatorLauncher  
<locator-launcher-options>
```

Specifically, you use the LocatorLauncher class API to run an embedded Locator service in Java application processes that you have created.

- **Co-located locator.** Manage a co-located locator at member startup or through the APIs:

- Use the gemfire.properties start-locator setting to start the locator automatically inside your GemFire member. See [gemfire.properties and gfsecurity.properties \(GemFire Properties\)](#) on page 851. The locator stops automatically when the member exits. The property has the following syntax:

```
#gemfire.properties  
start-locator=[address]port[,server={true|false},peer={true|false}]
```

Example:

```
#gemfire.properties  
start-locator=13489
```

- Use com.gemstone.gemfire.distributed.LocatorLauncher API to start the locator inside your code. Use the LocatorLauncher.Builder class to construct an instance of the LocatorLauncher, and then use the start() method to start a Locator service embedded in your Java application process. The other methods in the LocatorLauncher class provide status information about the locator and allow you to stop the locator.

## Check Locator Status

If you are connected to the distributed system in gfsh, you can check the status of a running Locator by providing the Locator name. For example:

```
gfsh>status locator --name=locator1
```

If you are not connected to a distributed system, you can check the status of a local Locator by providing the process ID, the Locator's hostname and port, or the Locator's current working directory. For example:

```
gfsh>status locator --pid=2986
```

or

```
gfsh>status locator --host=host1 --port=1035
```

or

```
gfsh>status locator --dir=<locator_working_directory>
```

where *<locator\_working\_directory>* corresponds to the local working directory where the locator is running.

If successful, the command returns the following information (with the JVM arguments that were provided at startup):

```
gfsh>status locator --dir=locator1
Locator in C:\vFabricGemFire70\Latest\locator1 on GemFireStymon[10334] as
locator1
is currently online.
Process ID: 3336
Uptime: 2 hours 15 seconds
GemFire Version: 7.0
Java Version: 1.6.0_26
Log File: C:\vFabricGemFire70\Latest\locator1\locator1.log
JVM Arguments:
...
```

## Stop the Locator

If you are connected to the distributed system in gfsh, you can stop a running locator by providing the locator name. For example:

```
gfsh>stop locator --name=locator1
```

If you are not connected to a distributed system, you can stop a local locator by specifying the locator's process ID or the locator's current working directory. For example:

```
gfsh>stop locator --pid=2986
```

or

```
gfsh>stop locator --dir=<locator_working_directory>
```

where *<locator\_working\_directory>* corresponds to the local working directory where the locator is running.



# Chapter 53

---

## vFabric GemFire Servers

A GemFire server is a vFabric GemFire process that runs as a long-lived, configurable member of a distributed system.

The GemFire server is used primarily for hosting long-lived data regions and for running standard GemFire processes such as the server in a client/server configuration. You can start and stop cache servers using the following methods:

- The easiest way to manage a GemFire The gfsh tool allows you to manage GemFire server processes from the command line.
- You can also start, stop and manage the GemFire servers through the `com.gemstone.gemfire.distributed.ServerLauncher` API. The `ServerLauncher` API can only be used for GemFire Servers that were started with `gfsh` or with the `ServerLauncher` class itself. See the JavaDocs for additional specifics on using the `ServerLauncher` API.



**Note:** The `ServerLauncher` API cannot be used reliably for Servers started using the deprecated GemFire shell script, `cacheserver`.

You must have `tools.jar` (included with the JDK) added to your CLASSPATH in order to manage servers using either `gfsh` or the `ServerLauncher` API.

### Server Configuration and Log Files

The `gfsh` server commands use a working directory for its configuration files and log files. These are the defaults and configuration options:

- Cache servers are configured like any other GemFire process, with `gemfire.properties` and `cache.xml` files. It is not programmable except through application plug-ins. Typically, you provide the `gemfire.properties` file, the `gfsecurity.properties` file (if you are using a separate, restricted access security settings file) and the `cache.xml` file in the cache server's working directory.
- For logging output, log file output defaults to `server_name.log` in the cache server's working directory. If you restart a server with the same server name, the existing `server_name.log` file is automatically renamed for you (for example, `server1-01-01.log` or `server1-02-01.log`). You can modify the level of logging details in this file by specifying a level in the `--log-level` argument when starting up the server.
- By default, the cache server will start in a subdirectory (named after the cache server) under the directory where `gfsh` is executed. This subdirectory is considered the current working directory. You can also specify a different working directory when starting the cache server in `gfsh`.

### Start the Cache Server

The startup syntax for cache servers in `gfsh` is:

```
start server --name=value [--assign-buckets(=value)?]
[--cache-xml-file=value] [--classpath=value] [--disable-default-server(=value)?]
```

```
[--enable-time-statistics(=value)?] [--force(=value)?] [--properties-file=value]
[--group=value][--license-application-cache=value]
[--license-data-management=value]
[--locators=value] [--log-level=value] [--mcast-address=value]
[--mcast-port=value]
[--memcached-port=value] [--rebalance(=value)?] [--server-bind-address=value]
[--server-port=value] [--statistic-archive-file=value] [--dir=value]
[--initial-heap=value] [--max-heap=value] [--J=value(,value)*]
```

See [start serverStarts a GemFire cache server.](#) for a detailed reference on all the parameters of the command.



**Note:** When both --max-heap and --initial-heap are specified during Server startup, additional GC parameters are specified internally by GemFire's Resource Manager. If you do not want the additional default GC properties set by the Resource Manager, then use the -Xms & -Xmx JVM options. See [Control Heap Use with the Resource Manager](#) on page 454 for more information.

These following gfsh start server start sequences use a single XML file for cache configuration, specify different incoming client connection ports, and use a multicast port for peer discovery:

```
gfsh>start server --name=server1 --mcast-port=10338 \
--cache-xml-file=../serverConfig/cache.xml --server-port=40404

gfsh>start server --name=server2 --mcast-port=10338
--cache-xml-file=../serverConfig/cache.xml --server-port=40405
```

Here, the multicast port and cache.xml file specifications are in a gemfire.properties file:

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

To start the cache server using this gemfire.properties file, you can enter the following command:

```
gfsh>start server --name=server1 \
--properties-file=D:\gfe\gemfire.properties
```

To start a cache server with an embedded JMX Manager, you can enter the following command:

```
gfsh>start server --name=<server-name> --J=-Dgemfire.jmx-manager=true \
--J=-Dgemfire.jmx-manager-start=true
```

## Check Server Status

If you are connected to the distributed system in gfsh, you can check the status of a running cache server by providing the server name. For example:

```
gfsh>status server --name=server1
```

If you are not connected to a distributed system, you can check the status of a local cache server by providing the process ID or the server's current working directory. For example:

```
gfsh>status server --pid=2484
```

or

```
gfsh>status server --dir=<server_working_directory>
```

where <server\_working\_directory> corresponds to the local working directory where the cache server is running.

If successful, the command returns the following information (with the JVM arguments that were provided at startup):

```
gfsh>status server --dir=server1
Server in C:\vFabricGemFire70\Latest\server1 on GemFireStymon[40404] as server1
```

```
is currently online.
Process ID: 2484
Uptime: 32 minutes 44 seconds
GemFire Version: 7.0
Java Version: 1.6.0_26
Log File: C:\vFabricGemFire70\Latest\server1\server1.log
JVM Arguments:
...

```

## Stop Server

If you are connected to the distributed system in gfsh, you can stop a running cache server by providing the server name. For example:

```
gfsh>stop server --name=server1
```

If you are not connected to a distributed system, you can stop a local cache server by specify the server's current working directory or the process ID. For example:

```
gfsh>stop server --pid=2484
```

or

```
gfsh>stop server --dir=<server_working_directory>
```

where <server\_working\_directory> corresponds to the local working directory where the cache server is running.

You can also use the gfsh shutdown command to shut down all cache servers in an orderly fashion. This is useful if you are using persistent regions. See [Shutting Down the System](#) on page 638 for more details.

## RHEL Only: Using the cacheserver Service

When you install vFabric GemFire from RPM, the RPM also installs a cacheserver service. This allow you to manage a default cache server with the following commands:

```
/sbin/service cacheserver start
/sbin/service cacheserver stop
/sbin/service cacheserver restart
```

This cacheserver service obtains its default configuration from  
`/etc/vmware/vfabric-gemfire/cacheserver.conf`.

Inside the configuration file, the following default properties are defined:

- Location of the cacheserver script:

```
CACHESEVER_SCRIPT=/opt/vmware/vmware-gemfire/vFabric_GemFire_70/bin/cacheserver
```

- Current working directory for the server:

```
CACHESEVER_WORK_DIR=/var/log/vmware/vfabric-gemfire/cacheserver
```

- Declarative cache configuration for the cache server:

```
CACHE_XML_FILE=/opt/vmware/vmware-gemfire/vFabric_GemFire_70/defaultConfigs/cache.xml
```

- Location of the gemfire.properties file for the cache server:

```
JAVA_OPTS="-DgemfirePropertyFile=/opt/vmware/vmware-gemfire/vFabric_GemFire_70/defaultConfigs/gemfire.properties"
```

You can modify this configuration to point your cacheserver service to a customized `gemfire.properties` file. In `gemfire.properties`, you can specify the `name` property, which will allow you to manage this cache server using the gfsh command-line utility.

Edit the `cacheserver.conf` file to configure the `cacheserver` service. You can also define your own additional `cacheserver` services. See `/etc/sysconfig/cacheserver` for instructions.

# Chapter 54

---

## Using Member Groups

vFabric GemFire allows you to organize your distributed system members into logical member groups.

The use of member groups in vFabric GemFire is optional. The benefit of using member groups is the ability to coordinate certain operations on members based on logical group membership. For example, by defining and using member groups you can:

- Alter a subset of configuration properties for a specific member or members. See [alter runtime](#) on page 674 in gfsh.
- Perform certain disk operations like disk-store compaction across a member group. See [Disk Store Commands](#) on page 660 for a list of commands.
- Manage specific indexes or regions across all members of a group.
- Start and stop multi-site (WAN) services such as gateway senders and gateway receivers across a member group.
- Deploy or undeploy JAR applications on all members in a group.
- Execute functions on all members of a specific group.

You define group names in the `groups` property of your member's `gemfire.properties` file.



**Note:** Any roles defined in the currently existing `roles` GemFire property will now be considered a group. If you wish to add membership roles to your distributed system, you should add them as member groups in the `groups` property. The `roles` property has been deprecated in favor of using the `groups` property.

To add a member to a group, add the name of a member group to the `gemfire.properties` file of the member prior to startup or you can startup a member in `gfsh` and pass in the `--group` argument at startup time.

A single member can belong to more than one group.

Member groups can also be used to organize members from either a client's perspective or from a peer member's perspective. See [Organizing Peers into Logical Member Groups](#) on page 174 and [Organizing Servers Into Logical Member Groups](#) on page 184 for more information. On the client side, you can supply the member group name when configuring a client's connection pool. Use the `<pool server-group>` element in the client's `cache.xml`.



# Chapter 55

---

## Deploying Application JARs to vFabric GemFire Members

You can dynamically deploy your application JAR files to specific members or to all members in your distributed system. GemFire automatically keeps track of JAR file versions; autoloads the deployed JAR files to the CLASSPATH; and auto-registers any functions that the JAR contains.

To deploy and undeploy application JAR files in vFabric GemFire, use the gfsh deploy or undeploy command. You can deploy a single JAR or multiple JARs (by either specifying the JAR filenames or by specifying a directory that contains the JAR files), and you can also target the deployment to a member group or multiple member group. For example, after connecting to the distributed system where you want to deploy the JARs, you could type at the gfsh prompt :

```
gfsh> deploy --jar=group1_functions.jar
```

This command deploys the group1\_functions.jar file to all members in the distributed system.

To deploy the JAR file to a subset of members, use the --group argument. For example:

```
gfsh> deploy --jar=group1_functions.jar --group=MemberGroup1
```

In the example it is assumed that you have already defined the member group that you want to use when starting up your members. See [Using Member Groups](#) on page 745 for more information on how to define member groups and add a member to a group.

To deploy all the JAR files that are located in a specific directory to all members:

```
gfsh> deploy --dir=libs/group1-libs
```

You can either provide a JAR file name or a directory of JARs for deployment, but you cannot specify both at once.

To undeploy all previously deployed JAR files throughout the distributed system:

```
gfsh> undeploy
```

To undeploy a specific JAR file:

```
gfsh> undeploy --jar=group1_functions.jar
```

To target a specific member group when all JAR files:

```
gfsh> undeploy --group=MemberGroup1
```

Only JAR files that have been previously deployed on members in the MemberGroup1 group will be undeployed.

To see a list of all deployed JARs in your distributed system:

```
gfsh> list deployed
```

To see a list of all deployed JARs in a specific member group:

```
gfsh> list deployed --group=MemberGroup1
```

Sample output:

Member	Deployed JAR	JAR Location
datanode1	group1_functions.jar	/usr/local/gemfire/deploy/vf.gf#group1_functions.jar#1
datanode2	group1_functions.jar	/usr/local/gemfire/deploy/vf.gf#group1_functions.jar#1

For more information on gfsh usage, see [Using the GemFire SHell \(gfsh\)](#) on page 641.

### Deployment Location for JAR Files

The system location where JAR files are written on each member is determined by the `deploy-working-dir` GemFire property configured for that member. For example, you could have the following configured in the `gemfire.properties` file for your member:

```
#gemfire.properties
deploy-working-dir=/usr/local/gemfire/deploy
```

This deployment location can be local or a shared network resource (such as a mount location) used by multiple members in order to reduce disk space usage. If you use a shared directory, you still need to deploy the JAR file on every member that you want to have access to the application, because deployment updates the CLASSPATH and auto-register functions.

### Versioning of JAR Files

When you deploy JAR files to a distributed system or member group, the JAR file is modified to indicate version information in its name. Each JAR filename is prefixed with `vf.gf#` and contains a version number at the end of the filename. For example, if you deploy `MyClasses.jar` five times, the filename is displayed as `vf.gf#MyClasses.jar#5` when you list all deployed jars.

When you deploy a new JAR file, the member receiving the deployment checks whether the JAR file is a duplicate, either because the JAR file has already been deployed on that member or because the JAR file has already been deployed to a shared deployment working directory that other members are also using. If another member has already deployed this JAR file to the shared directory (determined by doing a byte-for-byte compare to the latest version in its directory), the member receiving the latest deployment does not write the file to disk. Instead, the member updates the ClassPathLoader to use the already deployed JAR file. If a newer version of the JAR file is detected on disk and is already in use, the deployment is canceled.

When a member begins using a JAR file, the member obtains a shared lock on the file. If the member receives a newer version by deployment, the member releases the shared lock and tries to delete the existing JAR file in favor of the newer version. If no other member has a shared lock on the existing JAR, the existing, older version JAR is deleted.

### Automatic Class Path Loading

When a cache is started, the new cache requests that the latest versions of each JAR file in the current working directory be added to the ClassPathLoader. If a JAR file has already been deployed to the ClassPathLoader, the ClassPathLoader updates its loaded version if a newer version is found; otherwise, there is no change. If detected, older versions of the JAR files are deleted if no other member has a shared lock on them.

Undeploying a JAR file does not automatically unload the classes that were loaded during deployment. You need to restart your members to unload those classes.

When a cache is closed it requests that all currently deployed JAR files be removed from the ClassPathLoader.

If you are using a shared deployment working directory, all members sharing the directory should belong to the same member group. Upon restart, all members that share the same deployment working directory will deploy and autoload their CLASSPATH with any JARs found in the current working directory. This means that some members may load the JARs even though they are not part of the member group that received the original deployment.

### **Automatic Function Registration**

When you deploy a JAR file that contains a function (in other words, contains a class that implements the Function interface), the function is automatically registered through the `FunctionService.registerFunction` method. If another JAR file is deployed (either with the same JAR filename or another filename) with the same function, the new implementation of the function is registered, overwriting the old one. If a JAR file is undeployed, any functions that were auto-registered at the time of deployment are unregistered. Because deploying a JAR file that has the same name multiple times results in the JAR being un-deployed and re-deployed, functions in the JAR are unregistered and re-registered each time this occurs. If a function with the same ID is registered from multiple differently named JAR files, the function is unregistered if any of those JAR files are re-deployed or un-deployed.

During `cache.xml` load, the parameters for any declarables are saved. If functions found in a JAR file are also declarable, and have the same class name as the declarables whose parameters were saved after loading `cache.xml`, then function instances are created using those Parameters and are also registered. Therefore, if the same function is declared multiple times in the `cache.xml` with different sets of parameters, when the JAR is deployed a function is instantiated for each set of parameters. If any functions are registered using parameters from a `cache.xml` load, the default, no-argument function is not registered.



# Chapter 56

---

## Firewall Considerations

You can configure and limit port usage for situations that involve firewalls, for example, between client -server or server-server connections.

### Firewalls and Connections

Be aware of possible connection problems that can result from running a firewall on your machine.

vFabric GemFire 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.

As one example, firewalls may close connections to GemFire due to timeout settings. If a firewall senses no activity in a certain time period, it may close a connection and open a new connection when activity resumes, which can cause some confusion about which connections you have.

### Firewalls and Ports

Make sure your port settings are configured correctly for firewalls.

For a server, there are two different port settings you may need to be concerned with regarding firewalls:

- Port that the cache server listens on. This is configurable using the cache-server element in xml, on the CacheServer class in java, and as a command line option to the `gfsh start server` command.
- Locator port. GemFire clients can use the locator to automatically discover cache servers. The locator port is the same one that is configured for peer-to-peer messaging. The locator port is configurable in a command-line option to the `gfsh start locator` command.

For a client, you tell the client how to connect to the server using the pool options. In the client's pool configuration you can create a pool with either a list of server elements or a list of locator elements. For each element, you specify the host and port to connect to.

By default, GemFire clients and servers discover each other on a pre-defined port (40404) on the localhost.

Each gateway receiver uses a port to listen for incoming communication from one or more gateway senders communication between GemFire sites.

### Limiting Ephemeral Ports for Peer-to-Peer Membership

By default, GemFire assigns *ephemeral* ports, that is, temporary ports assigned from a designated range, which can encompass a large number of possible ports. When a firewall is present, the ephemeral port range usually must be limited to a much smaller number, for example six. If you are configuring P2P communications through a firewall, you must also set each the tcp port for each process and ensure that UDP traffic is allowed through the firewall.

### Properties for Firewall and Port Configuration

This table contains properties potentially involved in firewall behavior, with a brief description of each property. Click on a property name for a link to the gemfire.properties topic in the GemFire Reference section.

Gemfire Properties		
gemfire.properties	Setting	Definition
peer-to-peer config	<a href="#">gemfire.properties and gfsecurity.properties (GemFire Properties)</a>	Specifies whether sockets are shared by the system member's threads.
peer-to-peer config	<a href="#">gemfire.properties and gfsecurity.properties (GemFire Properties)</a>	The list of locators used by system members. The list must be configured consistently for every member of the distributed system.
peer-to-peer config	<a href="#">gemfire.properties and gfsecurity.properties (GemFire Properties)</a>	Address used to discover other members of the distributed system. Only used if mcast-port is non-zero. This attribute must be consistent across the distributed system.
peer-to-peer config	<a href="#">gemfire.properties and gfsecurity.properties (GemFire Properties)</a>	Port used, along with the mcast-address, for multicast communication with other members of the distributed system. If zero, multicast is disabled for member discovery and distribution.
peer-to-peer config	<a href="#">gemfire.properties and gfsecurity.properties (GemFire Properties)</a>	The range of ephemeral ports available for unicast UDP messaging and for TCP failure detection in the peer-to-peer distributed system.
peer-to-peer config	<a href="#">gemfire.properties and gfsecurity.properties (GemFire Properties)</a>	The TCP port to listen on for cache communications.

Server Configuration Properties		
gemfire.properties	Setting	Definition
cache server config	<a href="#">hostname-for-clients</a>	Hostname or IP address to pass to the client as the location where the server is listening.

Server Configuration Properties		
gemfire.properties	Setting	Definition
cache server config	max-connections	Maximum number of client connections for the server. When the maximum is reached, the server refuses additional client connections.
cache server config	port	Port that the server listens on for client communication.

Default Ports	
Port Name	Default Port
HTTP	8080
RMI	1099
RMI Server	0
Bridge Server	40404
Gateway Hub	1
Cache Server	40404
Multicast	10334
TCP	ephemeral port
Membership Port Range	1024 to 65535
Locator	no default

For more information, see the following sections:

- [How Client/Server Connections Work](#) on page 180
- [Socket Communication](#) on page 530
- [Controlling Socket Use](#) on page 520
- [Setting Socket Buffer Sizes](#) on page 531

## Locators, Gateway Receivers, and Ports

The ephemeral port range and TCP port range for locators must be accessible to members through the firewall. The full range of port values for gateway receivers also must be accessible from across the WAN.

Locators are used in the peer-to-peer cache to discover other processes. They can be used by clients to locate servers as an alternative to configuring clients with a collection of server addresses and ports.

Locators have a TCP/IP port that all members must be able to connect to. They also start a distributed system and so need to have their ephemeral port range and TCP port accessible to other members through the firewall.

Clients need only be able to connect to the locator's locator port. They don't interact with the locator's distributed system; clients get server names and ports from the locator and use these to connect to the servers. For more information, see [vFabric GemFire Locator Process](#) on page 737.

Each gateway receiver uses a single port to accept connections from gateway senders in other systems. However, the configuration of a gateway receiver specifies a range of possible port values to use. GemFire selects an available port from the specified range when the gateway receiver starts. Configure your firewall so that the full range of possible port values is accessible by gateway senders from across the WAN.

# Part 9

---

## Tools and Modules

*Tools and Modules* describes tools and modules associated with vFabric GemFire.

 **Note:** See also [Using the GemFire SHell \(gfsh\)](#) on page 641.

### Topics:

- [HTTP Session Management Modules](#)
- [Hibernate Cache Module](#)
- [Gemcached](#)
- [DiskConverterPre65to65](#)
- [GemFire Pulse](#)
- [DataBrowser](#)
- [Visual Statistics Display](#)



# Chapter 57

## HTTP Session Management Modules

The vFabric GemFire HTTP Session Management modules provide fast, scalable, and reliable session replication for HTTP servers without requiring application changes.

vFabric GemFire offers three HTTP session management modules:

- VMware vFabric GemFire HTTP Session Management Module 7.0 for tc Server
- VMware vFabric GemFire HTTP Session Management Module 7.0 for Tomcat
- VMware vFabric GemFire HTTP Session Management Module 7.0 for AppServers

These modules are included with the vFabric GemFire product distribution, and installation .zip files can be found in the tools/Modules directory of your product installation.



**Note:** The WAN configuration examples provided with the HTTP session management modules use the deprecated WAN configuration API. See [Configuring a Multi-site \(WAN\) System](#) on page 195 for information about the current WAN API.

### HTTP Session Management Quick Start

In this section you download, install, and set up the HTTP Session Management modules.

#### Quick Start Instructions

1. Download and install one of the application servers supported by vFabric GemFire.

Supported Application Server	Version	Download Location
tc Server	2.5	<a href="https://myvmware.com/web/vmware/info/slug/datacenter_downloads/vmware_vfabric_tc_server2_5">https://myvmware.com/web/vmware/info/slug/datacenter_downloads/vmware_vfabric_tc_server2_5</a>
tc Server	2.6	<a href="https://myvmware.com/web/vmware/info/slug/datacenter_downloads/vmware_vfabric_tc_server2_6">https://myvmware.com/web/vmware/info/slug/datacenter_downloads/vmware_vfabric_tc_server2_6</a>
tc Server	2.7	<a href="https://myvmware.com/web/vmware/info/slug/application_platform/vmware_vfabric_tc_server2_7">https://myvmware.com/web/vmware/info/slug/application_platform/vmware_vfabric_tc_server2_7</a>
tc Server	2.8	<a href="https://myvmware.com/web/vmware/info/slug/application_platform/vmware_vfabric_tc_server2_8">https://myvmware.com/web/vmware/info/slug/application_platform/vmware_vfabric_tc_server2_8</a>
Tomcat	6.0	<a href="http://tomcat.apache.org/download-60.cgi">http://tomcat.apache.org/download-60.cgi</a>
Tomcat	7.0	<a href="http://tomcat.apache.org/download-70.cgi">http://tomcat.apache.org/download-70.cgi</a>
WebLogic	11g (10.3.x)	<a href="http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html">http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html</a>
WebSphere	7, 8	<a href="http://www.ibm.com/developerworks/downloads/ws/was/">http://www.ibm.com/developerworks/downloads/ws/was/</a>
JBoss	5, 6, 7	<a href="http://www.jboss.org/jbossas/downloads/">http://www.jboss.org/jbossas/downloads/</a>



**Note:** This table lists all application server versions that have been tested with the GemFire HTTP Session Management modules. However, the generic HTTP Session Management Module for AppServers is implemented as a servlet filter and should work on any application server platform that supports the Java Servlet 2.4 specification.

2. If you have not done so already, click on "VMware vFabric GemFire" from the [VMware product download page](#) to download vFabric GemFire.
3. The HTTP Session Management Modules installation .zip files are located in the **tools/Modules** directory of the GemFire product download. Locate the .zip file for the HTTP Session Management Module that you wish to install. Unzip the appropriate HTTP Session Management Module into the specified directory for your application server:

Supported Application Server	Version	Module	Target Location for Module
tc Server	2.5	HTTP Session Management Module for tc Server	<tc Server root dir>/templates
tc Server	2.6	HTTP Session Management Module for tc Server	<tc Server root dir>/templates
tc Server	2.7	HTTP Session Management Module for tc Server	<tc Server root dir>/templates
tc Server	2.8	HTTP Session Management Module for tc Server	<tc Server root dir>/templates
Tomcat	6.0	HTTP Session Management Module for Tomcat	\$CATALINA_HOME
Tomcat	7.0	HTTP Session Management Module for Tomcat	\$CATALINA_HOME
WebLogic	11g (10.3.x)	HTTP Session Management Module for AppServers	<i>any directory</i>
WebSphere	7, 8	HTTP Session Management Module for AppServers	<i>any directory</i>
JBoss	5, 6, 7	HTTP Session Management Module for AppServers	<i>any directory</i>

4. Complete the appropriate set up instructions for your application server described in the following sections:
  - [Additional Quick Start Instructions for tc Server Module](#) on page 758
  - [Additional Quick Start Instructions for Tomcat Module](#) on page 759
  - [Additional Instructions for AppServers Module](#) on page 759

### Additional Quick Start Instructions for tc Server Module

These steps provide a basic starting point for using the tc Server module. For more configuration options, see [HTTP Session Management Module for vFabric tc Server](#) on page 765.

1. Navigate to the root directory of tc Server.
2. Create a GemFire instance using one of the provided templates and start the instance. For example:

```
prompt$ ./tcruntime-instance.sh create my_instance_name --template
```

```
gemfire-p2p
prompt$ ./tcruntime-ctl.sh my_instance_name start
```

This will create and run a GemFire instance using the peer-to-peer topology and default configuration values. Another GemFire instance on another system can be created and started in the same way.

If you need to pin your tc Server instance to a specific tc Server runtime version, use the --version parameter when creating the instance. See [Pinning a tc Server Runtime Instance to a Specific Version](#) on page 768 for details.

### **Additional Quick Start Instructions for Tomcat Module**

These steps provide a basic starting point for using the Tomcat module. For more configuration options, see [HTTP Session Management Module for Tomcat](#) on page 776.

1. Modify Tomcat's `server.xml` and `context.xml` files. Configuration is slightly different depending on the topology you are setting up and the version of Tomcat you are using.

For example, in a peer-to-peer configuration using Tomcat 7, you would add the following entry within the `<server>` element of `server.xml`:

```
<Listener className="com.gemstone.gemfire.modules.session.catalina.
    PeerToPeerCacheLifecycleListener"/>
```

and the following entry within the `<context>` tag in the `context.xml` file:

```
<Manager className="com.gemstone.gemfire.modules.session.catalina.
    Tomcat7DeltaSessionManager"/>
```

See [Setting Up the HTTP Module for Tomcat](#) on page 776 for additional instructions.

2. Restart the application server.

### **Additional Instructions for AppServers Module**

These steps provide a basic starting point for using the AppServers module with WebLogic, WebSphere or JBoss. For more configuration options, see [HTTP Session Management Module for AppServers](#) on page 786.



#### **Note:**

- The `modify_war` script, described below, relies on files within the distribution tree and should not be run outside of a complete distribution.
- The `modify_war` script is a bash script and does not run on Windows.

To set up the AppServers module, perform the following steps:

1. Run the `modify_war` script against an existing .war or .ear file to integrate the necessary components. The example below will create a configuration suitable for a peer-to-peer GemFire system, placing the necessary libraries into `WEB-INF/lib` for wars and `lib` for ears and modifying any `web.xml` files:

```
prompt$ bin/modify_war -w my-app.war -x
```

2. A new war file will be created called `session-my-app.war`. This file can now be deployed to the server.

### **Advantages of Using GemFire for Session Management**

The HTTP Session Management Module enables you to customize how you manage your session data.

Depending on your usage model, the HTTP Session Management Module enables you to accomplish the following tasks:

- Replicate session data across multiple peers.

- Partition data across multiple servers.
- Distribute session data across a WAN.
- Manage your session data in many other customizable ways.

Using GemFire for session management has many advantages:

- **tc Server integration**

The GemFire module offers clean integration into the tc Server environment with minimal configuration changes necessary.

- **Scalability**

Applications with a small number of frequently-accessed sessions can **replicate** session information on all members in the system. However, when the number of concurrent sessions being managed is large, data can be **partitioned** across any number of servers (either embedded within your application server process or managed by GemFire cache servers), which allows for **linear scaling**. Additionally, capacity can be **dynamically added or removed** in a running system and GemFire re-executes a non-blocking, rebalancing logic to migrate data from existing members to the newly added members. When the session state memory requirements exceed available memory, each partition host can **overflow to disk**.

- **Server-managed session state**

Session state can be managed independent of the application server cluster. This allows applications or servers to come and go without impacting session lifetimes.

- **Shared nothing cluster-wide persistence:** Session state can be persisted (and recovered) - invaluable for scenarios where sessions manage critical application state or have long lifetimes. GemFire uses a shared nothing persistence model where each member can continuously append to rolling log files without ever needing to seek on disk, providing very high disk throughput. When data is partitioned, the total disk throughput can come close to the aggregate disk transfer rates across each of the members storing data on disk.

- **Session deltas**

When session attributes are updated, only the updated state that is sent over the wire (to cache servers and to replicas). This provides fast updates even for large sessions. Session state is always managed in a serialized state on the servers, avoiding the need for the cache servers to be aware of the application classes.

- **Tiered caching**

Applications can configure a local or near cache for in-process caching of the most frequently used session state. This local cache delegates to a farm of cache servers where the entire session state is partitioned across any number of peer cache servers. The local cache can be configured to consume a certain percentage of the total heap available before least-recently used (LRU) eviction. This is a simpler and more effective way to manage LRU caches as opposed to alternate strategies based on count or memory size, which increase the risk of getting an "OutOfMemoryException".

- **Application server sizing**

Another aspect of tiered-caching functionality is that session replication can be configured so that session objects are stored external to the application server process. This allows the heap settings on the application server to be much smaller than they would otherwise need to be.

- **Replication to remote clusters**

The session state can be asynchronously replicated to/from multiple clusters across a WAN. This allows for disaster recovery (DR) architectures, among other benefits.

- **High availability (HA), disk-based overflow, synchronization to backend data store, other GemFire features**

All the popular GemFire features are available. For example: more than one synchronous copy of the session state can be maintained providing high availability (HA); the session cache can overflow to disk if the memory

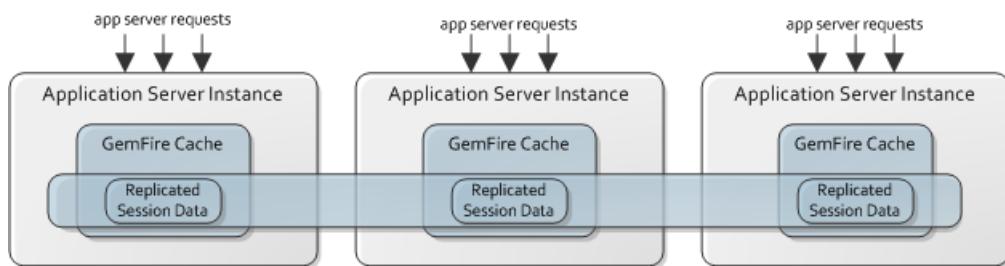
capacity in the cache server farm becomes insufficient; state information can be written to a backend database in a synchronous or asynchronous manner.

## Common Topologies for HTTP Session Management

Decide which topology is best for your usage. The module configuration process is slightly different for each topology.

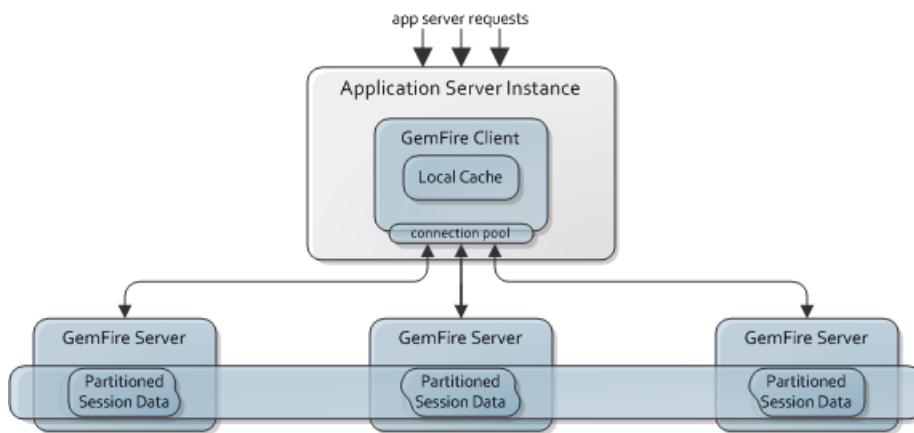
For information about vFabric GemFire purchase options as they relate to your desired configuration, refer to [How GemFire Manages Licenses](#) on page 52.

### Peer-to-Peer Configuration



In a peer-to-peer configuration, each GemFire instance within an application server contains its own cache. It communicates with other GemFire instances as peers (rather than clients and servers). Depending on the way the GemFire data region is defined, data is either replicated across all GemFire peers or it is partitioned so that each instance's cache contains a portion of the entire data set. By default, data is replicated. A peer-to-peer configuration is useful when multiple systems want fast access to all session data. This configuration is also the simplest one to set up and does not require any external processes.

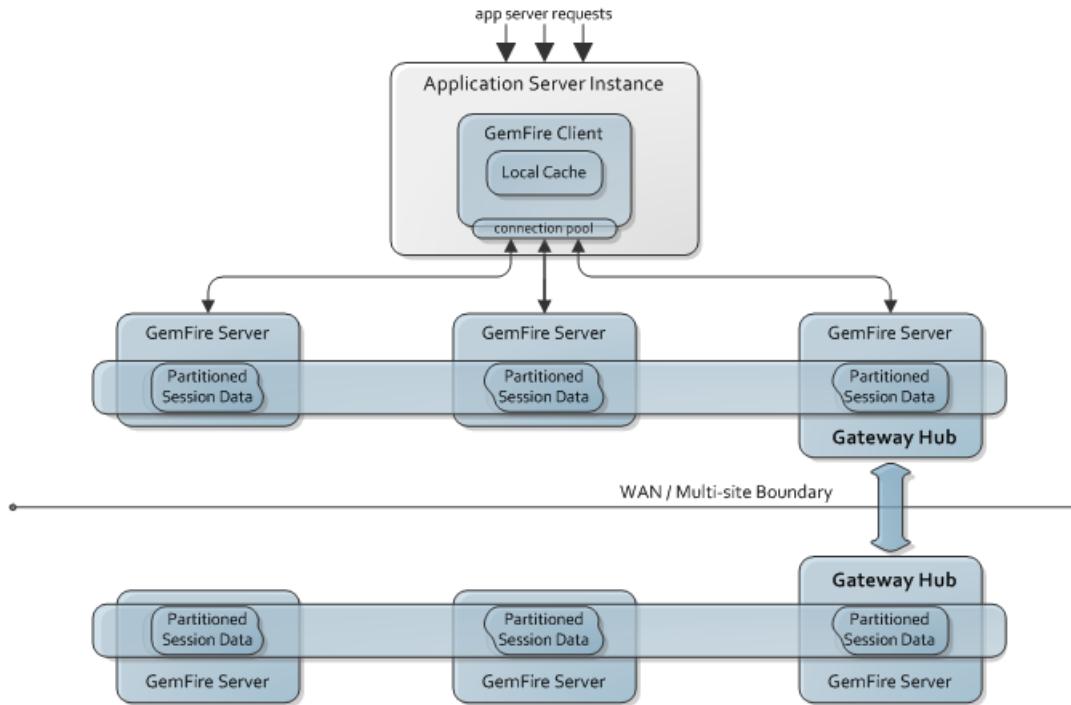
### Client/Server Configuration



In a client/server configuration, the Tomcat or tc Server instance operates as a GemFire client, which must communicate with one or more GemFire servers to acquire session data. The client maintains its own local cache and will communicate with the server to satisfy cache misses. A client/server configuration is useful when you want to separate the application server instance from the cached session data. In this configuration, you can

reduce the memory consumption of the application server since session data is stored in separate GemFire server processes.

### Multi-site (WAN-based) Configuration



With this configuration, distributed systems at different sites are loosely coupled through gateway system members (that is, gateway hubs). The distribution of session data between sites is seamless to the applications within each distributed system. If a system becomes unavailable, the rest of the cluster continues to operate. In a multi-site topology, each site has its own distinct distributed system. In each system, at least one member is chosen to act as a gateway that distributes data to and from the other sites. This configuration is useful when you are attempting to replicate session data across multiple sites, which can be valuable for handling complete site failovers. Load balancers such as [SpringSource ERS](#) can be configured with hot backups that access this remote HTTP session data.

Refer to [Topologies and Communication](#) on page 161 for more information on GemFire topologies.

## General Information on HTTP Session Management

This section provides information on sticky load balancers, session expiration, additional GemFire property changes, and more.

### Sticky Load Balancers

Typically, session replication will be used in conjunction with a load balancer enabled for sticky sessions. This involves, among other things, establishing a JVM route (`JVMRoute=value`). Refer to [SpringSource ERS](#) as a possible [load balancing](#) solution.

## Session Expiration

To set the session expiration value, you must change the `session-timeout` value specified in your application server's `WEB-INF/web.xml` file. This value will override the GemFire inactive interval specified by `maxInactiveInterval` within `context.xml`.

## Making Additional GemFire Property Changes

If you want to change additional GemFire property values, refer to instructions on manually changing property values as specified in the GemFire module documentation for Tomcat ([Changing the Default GemFire Configuration in the Tomcat Module](#) on page 779) and Application Servers ([Changing the Default GemFire Configuration in the AppServers Module](#) on page 793).

## Using the `gemfire` Command-line Tool

GemFire provides the `gemfire` command-line tool to perform basic administrative tasks. `gemfire.sh` (for Unix) and `gemfire.bat` (for Windows) should be located in the `/bin` subdirectory of the HTTP module distribution.



**Note:** The `gemfire` command-line tool is deprecated within GemFire (replaced by `gfh`) but can still be used with the HTTP modules.

## Module Version Information

To acquire GemFire module version information, look in the web server's log file for the following message:

```
Nov 8, 2010 12:12:12 PM
com.gemstone.gemfire.modules.session.catalina.AbstractCacheLifecycleListener
lifecycleEvent
INFO: Initializing GemFire Modules Version 1.1
```

## Session State Log Files

Several log files are written by the various parts of the session management code.

- `catalina.log`. Log file written by the tc server
- `cacheserver.log`. Log file written by the GemFire Cache Server.
- `gemfire_modules.log`. Log file written by the GemFire Cache Client.

## Adding FINE Debug Logging to `catalina.log`

To add GemFire-specific FINE logging to the `catalina.log` file, perform the following steps:

1. Add the following lines to your `logging.properties` file:

```
com.gemstone.gemfire.modules.session.catalina.DeltaSessionManager.level
= FINE
com.gemstone.gemfire.modules.session.catalina.DeltaSession.level = FINE
```

2. Add the following line to the tc Server startup properties:

```
-Djava.util.logging.config.file=/path/to/logging.properties
```

where `/path/to/logging.properties` corresponds to the location of the `logging.properties` file that you configured in step 1.

These configurations will add FINE logging to the catalina.DATE.log file. The following is an example of FINE logging:

```
06-Sep-2011 15:59:47.250 FINE
com.gemstone.gemfire.modules.session.catalina.DeltaSessionManager.start
DeltaSessionManager[container=StandardEngine[Catalina].StandardHost[localhost].StandardContext[/manager];
regionName=gemfire_modules_sessions; regionAttributesId=PARTITION_REDUNDANT ]:
Starting

06-Sep-2011 15:59:47.254 FINE
com.gemstone.gemfire.modules.session.catalina.DeltaSessionManager.registerJvmRouteBinderValve
DeltaSessionManager[container=StandardEngine[Catalina].StandardHost[localhost].StandardContext[/manager];
regionName=gemfire_modules_sessions; regionAttributesId=PARTITION_REDUNDANT ]:
Registering JVM route binder valve

06-Sep-2011 15:59:47.351 FINE
com.gemstone.gemfire.modules.session.catalina.ClientServerSessionCache.createOrRetrieveRegion
Created session region:
com.gemstone.gemfire.internal.cache.LocalRegion[path='/gemfire_modules_sessions';
scope=LOCAL';dataPolicy=EMPTY; gatewayEnabled=false]
```

### Add Session State Logging to cacheserver.log

To add session-state-specific logging to the cacheserver.log file, add the following property to the catalina.properties file for the tc Server instance:

```
gemfire-cs.enable.debug.listener=true
```

Adding this configuration will print logging in the cacheserver.log such as the following:

```
[info 2011/09/06 15:18:27.749 PDT <ServerConnection on port 40404 Thread
3> tid=0x32] DebugCacheListener: Received
CREATE for key=5782ED83A3D9F101BBF8D851CE4E798E;
value=DeltaSession[id=5782ED83A3D9F101BBF8D851CE4E798E;
sessionRegionName=gemfire_modules_sessions; operatingRegionName=unset]

[info 2011/09/06 15:18:27.769 PDT <ServerConnection on port 40404 Thread
3> tid=0x32] DebugCacheListener: Received UPDATE
for key=5782ED83A3D9F101BBF8D851CE4E798E;
value=DeltaSession[id=5782ED83A3D9F101BBF8D851CE4E798E;
sessionRegionName=gemfire_modules_sessions; operatingRegionName=unset]

[info 2011/09/06 15:19:36.729 PDT <Timer-2> tid=0x24] DebugCacheListener:
Received EXPIRE_DESTROY for
key=5782ED83A3D9F101BBF8D851CE4E798E
```

### Add Debug Logging to cacheserver.log

To add fine-level logging to the GemFire Cache Server, add the 'log-level' property to the cacheserver start script. For example:

```
./cacheserver.sh start -dir=server1
cache-xml-file=../conf/cache-server.xml log-level=fine
```

This will add fine-level logging to the cacheserver.log file.



**Note:** This will help debug GemFire server issues, but it adds a lot of logging to the file.

## Add Debug Logging to gemfire\_modules.log

To add fine-level logging to the GemFire Cache Client, add the 'log-level' property to the Listener element in the tc Server of Tomcat server.xml file. For example:

```
<Listener log-level="fine"
cache-xml-file="${gemfire-cs.cache.configuration.file}"
className="com.gemstone.gemfire.modules.session.catalina.ClientServerCacheLifecycleListener"
criticalHeapPercentage="${gemfire-cs.critical.heap.percentage}"
evictionHeapPercentage="${gemfire-cs.eviction.heap.percentage}"
log-file="${gemfire-cs.log.file}"
statistic-archive-file="${gemfire-cs.statistic.archive.file}"
statistic-sampling-enabled="${gemfire-cs.statistic.sampling.enabled}"/>
```

This will add fine-level logging to the file defined by the \${gemfire-cs.log.file} property. The default log file name is gemfire\_modules.log.



**Note:** This will help debug GemFire client issues, but it adds a lot of logging to the file.

## HTTP Session Management Module for vFabric tc Server

This section describes how to set up and use the HTTP session management module with tc Server templates.

If you would prefer to manually change the server.xml and context.xml files rather than use tc Server templates, refer to [HTTP Session Management Module for Tomcat](#) on page 776.

### Installing the HTTP Module for tc Server

This topic describes how to install the HTTP session management module with tc Server templates.

1. If you do not already have tc Server, download and install the product from the [vFabric tc Server download page](#). These instructions require **tc Server 2.1** or later. (If you are using an older version of tc Server, you should set up the plug-in without templates: [HTTP Session Management Module for Tomcat](#) on page 776.)
2. If you have not already downloaded and installed vFabric GemFire 7.0, download and install vFabric GemFire 7.0 from the [vFabric GemFire download page](#) as described in [Installing vFabric GemFireInstall or update vFabric GemFire, and configure your environment to run it.](#)

The HTTP Session Management Module for tc Server is included in the vFabric GemFire 7.0 installation package. After you install vFabric GemFire, you can find the module in the following location:

```
<product_dir>/tools/Modules/vFabric_GemFire_Modules-7.X.X.X-tcServer.zip
```

where <product\_dir> is the location where you installed GemFire and 7.X.X.X corresponds to the version of the HTTP Session Management Module you are installing.

3. Unzip the module into the vFabric tc Server \$CATALINA\_HOME/templates directory so that it creates gemfire-p2p and gemfire-cs subdirectories within the tc Server templates directory.
4. Copy the following jar files to the lib directory of your tc server (\$CATALINA\_HOME/lib):
  - <product\_dir>/lib/gemfire.jar
  - <product\_dir>/lib/antlr.jar
  - <product\_dir>/lib/spring-core-3.1.1.RELEASE.jar
  - <product\_dir>/lib/spring-shell-1.0.0.RELEASE.jar



**Note:** The GemFire evaluation license allows you to create up to three GemFire processes. To use GemFire with additional processes, refer to [How GemFire Manages Licenses](#) on page 52.

## Setting Up the HTTP Module for tc Server

A simple configuration requires you to start a tc Server instance with the appropriate tc Server template, depending on your preferred topology.

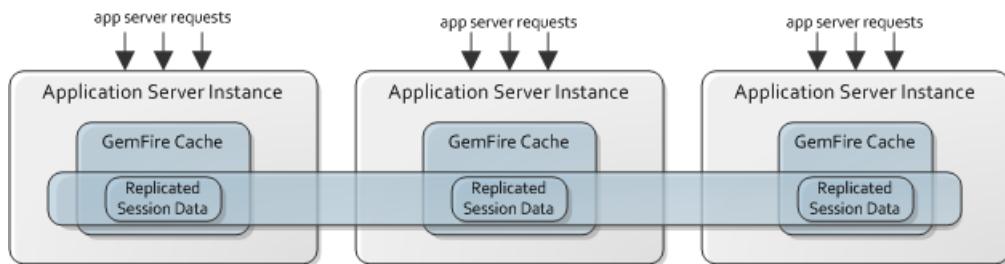
- [Peer-to-Peer Setup](#) on page 766
- [Client/Server Setup](#) on page 766
- [Pinning a tc Server Runtime Instance to a Specific Version](#) on page 768
- [Starting the Application Server](#) on page 768
- [Verifying That GemFire Started](#) on page 768
- [Configuring Non-Sticky Session Replication](#) on page 768

Refer to [Common Topologies for HTTP Session Management](#) on page 761 for more information.



**Note:** In the procedures that follow, you may be required to log in as root or use sudo to run the commands in Unix, especially if you installed vFabric tc Server from RPM using yum.

### Peer-to-Peer Setup



To run GemFire in a peer-to-peer configuration, create a tc Server instance using the supplied "gemfire-p2p" template:

In Unix:

```
prompt$ ./tcruntime-instance.sh create my_instance_name --template
gemfire-p2p
```

In Windows:

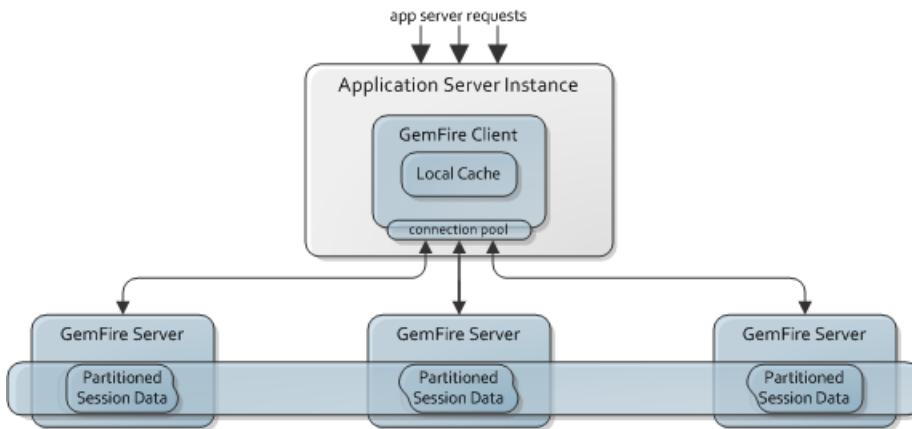
```
prompt> tcruntime-instance.bat create my_instance_name --template
gemfire-p2p
```

If you do not make any further configuration changes, peers attempt to locate each other using a default multicast channel. You can change the multicast port, as shown in [Using a Different Multicast Port](#) on page 770.

Alternatively, peers can discover each other using locators, which provide both discovery and load balancing services. Refer to [Peer-to-peer Configuration Using Locators](#) on page 770 for more information.

If you need to pin your tc Server instance to a specific version of the tc Runtime, specify the `--version` parameter in your command to create the tc Server instance. See [Pinning a tc Server Runtime Instance to a Specific Version](#) on page 768

### Client/Server Setup



When you run GemFire in a client/server configuration, the application server (tc Server) operates as a GemFire client. Create a tc Server instance using the supplied "gemfire-cs" template:

In Unix:

```
prompt$ ./tcruntime-instance.sh create my_instance_name --template
gemfire-cs
```

In Windows:

```
prompt> tcruntime-instance.bat create my_instance_name --template
gemfire-cs
```

If you need to pin your tc Server instance to a specific version of the tc Runtime, specify the `--version` parameter in your command to create the tc Server instance. See [Pinning a tc Server Runtime Instance to a Specific Version](#) on page 768

Because the application server operates as a GemFire client in this configuration, you must manually launch a GemFire cache server. You can do this with the following script (which should be installed into the `bin` subdirectory for the applicable template):

In Unix:

```
prompt$ ./cacheserver.sh start
```

In Windows:

```
prompt> cacheserver.bat start
```



**Note:** If you plan to use the deprecated `cacheserver` script that comes with the standalone GemFire product instead of the `cacheserver` script that is provided with the module, you must manually add the following items to the classpath: `INSTANCE_DIR/lib/servlet-api.jar`, `INSTANCE_DIR/lib/catalina.jar`, `INSTANCE_DIR/bin/tomcat-juli.jar`, `INSTANCE_DIR/conf` (where '`INSTANCE_DIR`' is the location of the tc Server instance you created with the GemFire template). These items are automatically added to the classpath by the supplied `cacheserver.sh` (or `cacheserver.bat`) script that comes with the tc Server HTTP Session Management module.

With no further configuration changes, the client and server attempt to locate each other on the same system using a default port (40404). Although this is useful for testing purposes, you would not normally deploy a client/server configuration in this way. Refer to [Client/Server Configuration Using Locators](#) on page 771 for information on creating and using locators, which provide both discovery and load balancing services.



**Note:** Because the client/server configuration uses a local client cache by default, it is important to set up your application server for sticky sessions so that application clients access the same application server

across a session interval. See [General Information on HTTP Session Management](#) on page 762 for more information.

### Pinning a tc Server Runtime Instance to a Specific Version

You can pin a specific version of the tc Server runtime when creating your tc Server instance. To specify the version, use the `--version` parameter when executing the `tcruntime-instance.sh` or `tcruntime-instance.bat` script. For example:

In Unix:

```
prompt$ ./tcruntime-instance.sh create --version 6.0.32.A.RELEASE \
my_instance_name --template gemfire-cs
```

In Windows:

```
prompt> tcruntime-instance.bat create my_instance_name --template gemfire-cs
```

When you pin the tc Server runtime instance to a specific version, the created instance will always use the specified version, even if you install a more recent version of the tc Server runtime. If you do not specify the `--version` parameter, the tc Server runtime is automatically pinned to the highest version that you have installed.

To determine the list of available versions, search for `INSTALL_DIR/tomcat-XXX` directories where `XXX` follows a pattern such as `6.0.32.A.RELEASE`.

### Starting the Application Server

Once you've created a tc Server instance, you are ready to start the instance.

In Unix:

```
prompt$ ./tcruntime-ctl.sh my_instance_name start
```

In Windows:

```
prompt> ./tcruntime-ctl.bat my_instance_name start
```

Refer to the [tc Server](#) documentation for more information. Once started, GemFire will automatically launch within the application server process.



**Note:** GemFire session state management provides its own clustering functionality. If you are using GemFire, you should NOT turn on Tomcat clustering as well.

### Verifying That GemFire Started

You can verify that GemFire has successfully started by inspecting the Tomcat log file. For example:

```
Nov 8, 2010 12:12:12 PM
com.gemstone.gemfire.modules.session.catalina.ClientServerCacheLifecycleListener
createOrRetrieveCache
INFO: Created GemFireCache[id = 2066231378; isClosing = false;
      created = Mon Nov 08 12:12:12 PDT 2010; server = false;
      copyOnRead = false; lockLease = 120; lockTimeout = 60]
```

Information is also logged within the GemFire log file, which by default is named "gemfire\_modules.log".

### Configuring Non-Sticky Session Replication

Some situations require that sessions be 'non-sticky', which means that client requests can be directed to any server in a cluster of application servers instead of the same server each time. To achieve this, you must configure your deployment as described for the following topologies.

#### Peer-to-Peer

For peer-to-peer topologies, apply the following settings to enable non-sticky sessions to work correctly:

- Java system property `gemfire.loadClassOnEveryDeserialization=true`. Set this property by editing the `bin/setenv.sh` file.
- `prefer.deserialized.form=false`. Set this property in `conf/catalina.properties`.

### Client-server

For client-server topologies, only the Java system property

`gemfire.loadClassOnEveryDeserialization=true` must be set, in the `bin/setenv.sh` file.



**Note:** Non-sticky sessions cause a performance impact because sessions need to be re-created every time a request hits a different server. This may not be noticeable when the session attributes are small, but may become more evident as the session attributes increase in size and/or number.

### Changing the Default GemFire Configuration in the tc Server Module

By default, the tc Server HTTP module will run GemFire automatically with pre-configured settings. You can change these GemFire settings.

Here are the default GemFire settings:

- GemFire peer-to-peer members are discovered using a default multicast channel.
- GemFire locators are not used for member discovery.
- The region name is set to `GemFire_modules_sessions`.
- The cache region is replicated for peer-to-peer configurations and partitioned (with redundancy turned on) for client/server configurations.
- GemFire clients have local caching turned on and when the local cache needs to evict data, it will evict least-recently-used (LRU) data first.



**Note:** On the application server side, the default inactive interval for session expiration is set to 30 minutes. To change this value, refer to [Session Expiration](#) on page 763.

However, you may want to change this default configuration. For example, you might want to change the region from replicated to partitioned, or to use locators for discovery. To change GemFire settings, use the `--interactive` command line argument when creating your tc Server instance.

In Unix:

```
prompt$ ./tcruntime-instance.sh create my_instance_name --template
gemfire-p2p
    --interactive
prompt$ ./tcruntime-instance.sh create my_instance_name --template
gemfire-cs
    --interactive
```

In Windows:

```
prompt> tcruntime-instance.bat create my_instance_name --template
gemfire-p2p
    --interactive
prompt> tcruntime-instance.bat create my_instance_name --template
gemfire-cs
    --interactive
```

In interactive mode, you will be prompted to specify a series of property values. Hit <return> for any property where you would like to use the default value.

```
Creating instance 'my_instance_name' ...
Using separate layout
Please enter a value for 'gemfire.maximum.vm.heap.size.mb'. Default '512':
```

```
Please enter a value for 'gemfire.initial.vm.heap.size.mb'. Default '512':  
...  
...
```

After responding to approximately 20 prompts, you should see the following line:

```
Instance created.
```



**Note:** You cannot override region attributes on the cache server when using the HTTP Session Management Module. You must place all region attribute definitions in the region attributes template that you customize in tc Server. See [Overriding Region Attributes](#) on page 797 for more information.

For information on setting up your instance for the most common types of configurations, refer to the sections below. For more information about each interactive prompt, refer to [Interactive Configuration Reference for the tc Server Module](#) on page 774.

### Using a Different Multicast Port

For a GemFire peer-to-peer member to communicate on a different multicast port than the default (10334), answer the following question in the tc Server HTTP module's interactive mode:

```
Please enter the port used by GemFire members to discover each other using  
multicast networking. Default '10334': 10335
```

This example changes the multicast port to 10335.

You typically *should* change the multicast port so that you don't inadvertently join another GemFire cluster on the same network.

### Overriding Region Attributes

When using the HTTP Session Management Module, you cannot override region attributes directly on the cache server. You must place all region attribute definitions in the region attributes template that you customize within tc Server. For example, to specify a different name for the region's disk store, you could add the new disk-store-name specification to the region attributes template and then reference the template on the cache server.

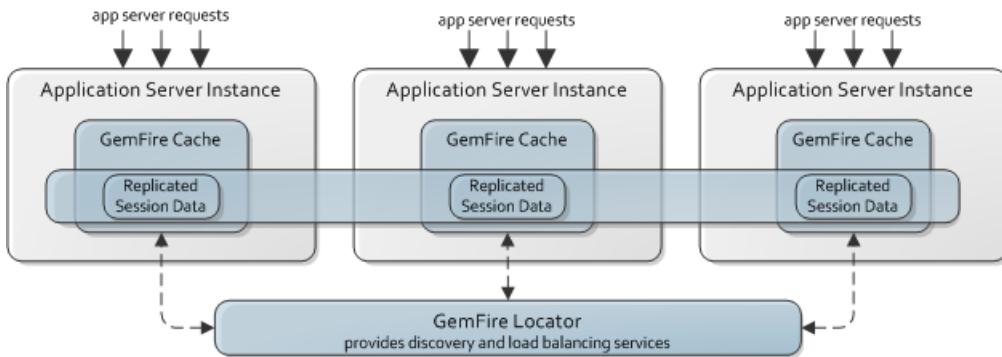
```
<region-attributes id="MY_SESSIONS"  
refid="PARTITION_REDUNDANT_PERSISTENT_OVERFLOW"  
disk-store-name="mystore">  
...  
</region-attributes>
```

Then on the cache server side, reference the modified region attributes template to allow the region to use the disk-store-name attribute:

```
<region name="gemfire_modules_sessions" refid="MY_SESSIONS" />
```

### Peer-to-peer Configuration Using Locators

By default, GemFire peers discover each other using multicast communication on a known port. Alternatively, you can configure member discovery using one or more dedicated locators. A locator provides both discovery and load balancing services.



To run GemFire with one or more locators, turn off multicast discovery (by setting it to 0) and specify the list of locators.

```
Please enter the port used by GemFire members to discover each other using
multicast networking. Default '10334': 0
```

```
Please enter the list of locators used by GemFire members to discover
each other.
```

```
The format is a comma-separated list of host[port]. Default ' ':
hostname[10334]
```

This example turns off multicast discovery and uses a locator at hostname on port 10334.

You must be sure when you specify a locator that you specifically launch a locator correctly at this location. You can do this using the `gemfire` command-line tool found in the `bin` subdirectory of the HTTP module distribution.

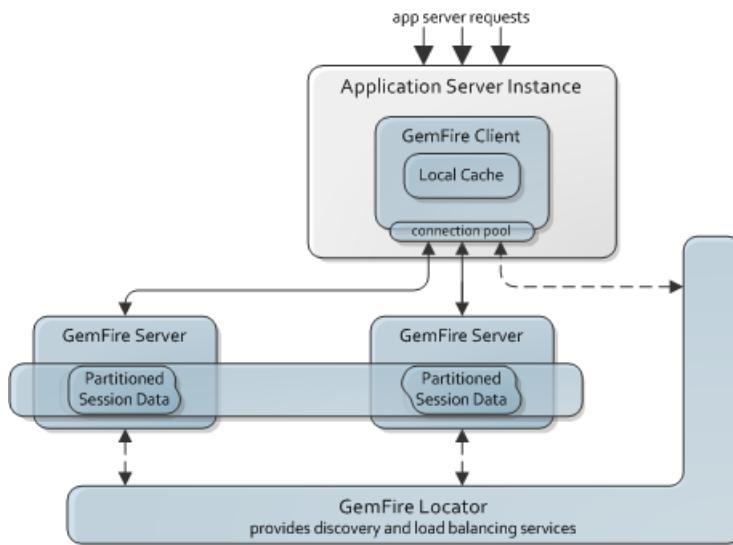
```
In Unix:
./gemfire.sh start-locator -port=10334
```

```
In Windows:
gemfire.bat start-locator -port=10334
```

This example starts a locator that listens on port 10334.

### **Client/Server Configuration Using Locators**

By default, GemFire clients and servers discover each other on a pre-defined port on the localhost. This is not typically the way you would deploy a client/server configuration. A preferred solution is to use one or more dedicated locators. A locator provides both discovery and load balancing services.



To run GemFire with one or more locators instead of a multicast port, add locator information to the GemFire `cache-client.xml` file in the `conf` subdirectory of the applicable tc Server instance.

```
<pool name="sessions">
  <locator host="hostname" port="10334"/>
</pool>
```

This example tells the GemFire client (which is the process running the tc Server application server) to use a locator at `hostname` on port 10334.

You must now launch a locator correctly at this location. You can do this using the `gemfire` command-line tool found in the `bin` subdirectory of the HTTP module distribution.

```
./gemfire start-locator -port=10334
```

This example starts a locator that listens on port 10334.

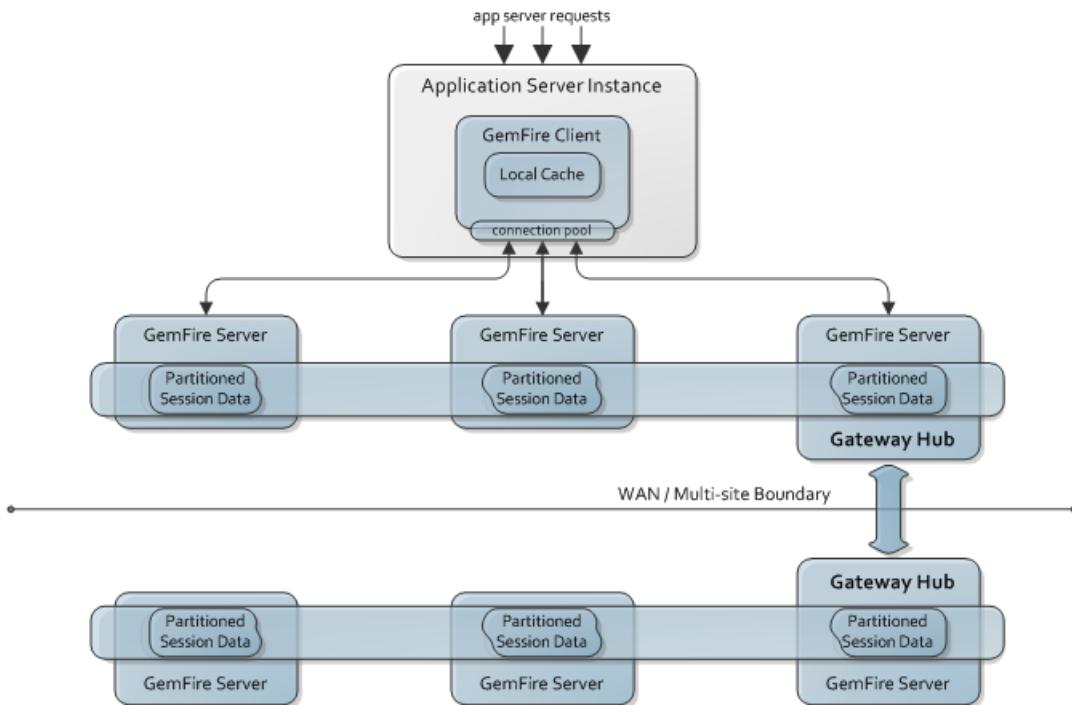
You also need to tell the GemFire server to use a locator when you launch the server. You can do this through a command-line argument when you start up the server with the script provided in the `bin` subdirectory wherever you installed the GemFire HTTP module:

```
In Unix:
prompt$ ./cacheserver.sh start locators=hostname[port]

In Windows:
prompt> cacheserver.bat start locators=hostname[port]
```

If you are running more than one locator, use a comma-separated list of locators.

## **Multi-site Setup**



For information on when to use this topology, refer to [Common Topologies for HTTP Session Management](#) on page 761. Setting up a gateway for multi-site communication requires a few configuration changes. You should first create an instance in interactive mode and respond to the following prompt by typing true:

```
Please specify whether session modifications should be replicated across
the WAN.
Default 'false': true
```

After creating the tc Server instance, you will need to modify the GemFire configuration file (either `cache-peer.xml` for a peer-to-peer configuration or `cache-server.xml` for a client/server configuration) in the `conf` subdirectory of the tc Server module instance that will operate as the gateway hub. For example, if you were setting up a gateway between a site named "SITE1" and a site named "SITE2", this is how you would set up the information for site 1:

```
<gateway-hub id="SITE1" port="22222" socket-buffer-size="256000">
    <gateway id="SITE2_CLUSTERA" socket-buffer-size="256000">
        <gateway-endpoint id="SITE2_CLUSTERA_server1" host="site2_hostname"
            port="22222" />
    </gateway>
</gateway-hub>
```

And at site 2, you would need to set up another gateway hub that communicates with site 1. Notice that the endpoint for site 1 references information about site 2 while the endpoint for site 2 references information about site 1.

```
<gateway-hub id="SITE2" port="22222" socket-buffer-size="256000">
    <gateway id="SITE1_CLUSTERA" socket-buffer-size="256000">
        <gateway-endpoint id="SITE1_CLUSTERA_server1" host="site1_hostname"
            port="22222" />
    </gateway>
</gateway-hub>
```



**Note:** To successfully run a configuration using a multi-site topology, you must start at least one server and one client in **each** site before creating or accessing your session. If a GemFire client is not started in one site, the region data containing the session information will not get initialized. This means that a web server must be started on **each** site before creating or accessing session data.

See [Multi-site \(WAN\) Configuration](#) on page 189 for more information.

### Interactive Configuration Reference for the tc Server Module

This section describes each prompt when entering into interactive configuration mode of the GemFire HTTP Session Management Module for tc Server.

See [Changing the Default GemFire Configuration in the tc Server Module](#) on page 769 for specific pointers on how to use the interactive mode for common configuration changes.

```
Please enter a value for 'gemfire.maximum.vm.heap.size.mb'. Default '512':  
Please enter a value for 'gemfire.initial.vm.heap.size.mb'. Default '512':  
Please enter a value for 'gemfire.cms.initiating.heap.percentage'. Default '50':
```

The above properties allow you to fine-tune your JVM heap and garbage collector. For more information, refer to [Heap Use and Management](#) on page 453.

```
Please specify whether to enable a GemFire listener that logs session  
create,  
update, destroy and expiration events. Default 'false':
```

The above property determines whether a debug cache listener is added to the session region. When true, info-level messages are logged to the GemFire log when sessions are created, updated, invalidated, or expired.

```
Please specify whether session modifications should be replicated across  
the WAN.  
Default 'false':
```

The above property determines whether sessions are replicated across the gateway. When true, a GatewayHub must be configured, as discussed in "Multi-site Setup".

With the gemfire-p2p template:

```
Please specify whether to maintain a local GemFire cache. Default 'false':
```

With the gemfire-cs template:

```
Please specify whether to maintain a local GemFire cache. Default 'true':
```

The above property determines whether a local cache is enabled; if this parameter is set to true, the app server load balancer should be configured for sticky session mode.

With the gemfire-p2p template:

```
Please enter the id of the attributes of the GemFire region used to cache  
sessions.  
Default 'REPLICATE':
```

With the gemfire-cs template:

```
Please enter the id of the attributes of the GemFire region used to cache  
sessions.  
Default 'PARTITION_REDUNDANT':
```

The above property determines the ID of the attributes for the cache region; possible values include PARTITION, PARTITION\_REDUNDANT, PARTITION\_PERSISTENT, REPLICATE, REPLICATE\_PERSISTENT, and any other region shortcut that can be found in [Region Shortcuts and Custom Named Region Attributes](#) on page

150. When using a partitioned region attribute, it is recommended that you use PARTITION\_REDUNDANT (rather than PARTITION) to ensure that the failure of a server does not result in lost session data.

```
Please enter the name of the GemFire region used to cache sessions.  
Default 'gemfire_modules_sessions':
```

The above property determines the GemFire region name.

```
Please enter the port that Tomcat Shutdown should listen on. Default '-1':  
  
Please enter the port that the JMX socket listener should listen on.  
Default '6969':
```

The above properties are application server properties.

```
Please enter a value for 'bio.http.port'. Default '8080':  
Please enter a value for 'bio.https.port'. Default '8443':
```

tc Server requires information about connector ports. bio.http.port is the http port for tc Server and bio.https.port is the secure http port for tc Server.

With the gemfire-p2p template:

```
Please enter the name of the GemFire cache configuration file.  
Default 'cache-peer.xml':
```

With the gemfire-cs template:

```
Please enter the name of the GemFire cache configuration file.  
Default 'cache-client.xml':
```

You can change the name of the cache configuration file with the above property. If you do change this value, be sure to include an xml file by that name in the conf subdirectory.

```
Please enter the percentage of heap at which updates to the cache are refused.  
0.0 means disabled. Default '0.0':  
Please enter the percentage of heap at which sessions will be evicted from the local cache. Default '80.0':
```

The above properties allow you to control the critical and eviction watermarks for the heap. By default, the critical watermark is disabled (set to 0.0) and the eviction watermark is set to 80%.

Applicable to the gemfire-p2p template ONLY:  
Please enter the list of locators used by GemFire members to discover each other.  
The format is a comma-separated list of host[port]. Default ' ':

The above property specifies the list of locators. For more information on locators, refer to [Peer-to-peer Configuration Using Locators](#) on page 770.

```
Please enter the name of the file used to log GemFire messages.  
Default 'gemfire_modules.log':
```

The above property determines the filename for the GemFire log file.

Applicable to the gemfire-p2p template ONLY:  
Please enter the port used by GemFire members to discover each other using multicast networking. Default '10334':

The above property specifies the port used for multicast discovery. When using locators, this value should be 0. For more information on multicast discovery, refer to [Using a Different Multicast Port](#) on page 770.

Applicable to the gemfire-p2p template ONLY:  
Please specify whether to rebalance the GemFire cache at startup.  
Default 'false':

This property allows you to rebalance a partitioned GemFire cache when a new GemFire peer starts up.

```
Please enter the name of the file used to store GemFire statistics.  
Default 'gemfire_modules.gfs':
```

The above property determines the filename for the GemFire statistics file.

```
Please specify whether GemFire statistic sampling should be enabled.  
Default 'false':
```

The above property determines whether statistics sampling should occur. See [Statistics](#) on page 551 for more information.

## HTTP Session Management Module for Tomcat

You set up and use the module by modifying the Tomcat's `server.xml` and `context.xml` files.

For instructions specific to SpringSource tc Server templates, refer to [HTTP Session Management Module for vFabric tc Server](#) on page 765.

### Installing the HTTP Module for Tomcat

This topic describes how to install the HTTP session management module for Tomcat.

1. If you do not already have Tomcat installed, download Tomcat 6.0 or 7.0 from the [Apache Website](#).
2. If you have not already downloaded and installed vFabric GemFire 7.0, download and install vFabric GemFire 7.0 from the [vFabric GemFire download page](#) as described in [Installing vFabric GemFireInstall or update vFabric GemFire, and configure your environment to run it](#).

The HTTP Session Management Module for Tomcat is included in the vFabric GemFire 7.0 installation package. After you install vFabric GemFire, you can find the module in the following location:

```
<product_dir>/tools/Modules/vFabric_GemFire_Modules-7.X.X-X-Tomcat.zip
```

where `<product_dir>` is the location where you installed GemFire and `7.X.X-X` corresponds to the version of the module you are installing.

3. Unzip the module into the `$CATALINA_HOME` directory or wherever you installed the application server.
4. Copy the following JAR files to the lib directory of your Tomcat server (`$CATALINA_HOME/lib`).
  - `<product_dir>/lib/gemfire.jar`
  - `<product_dir>/lib/antlr.jar`
  - `<product_dir>/lib/spring-core-3.1.1.RELEASE.jar`
  - `<product_dir>/lib/spring-shell-1.0.0.RELEASE.jar`

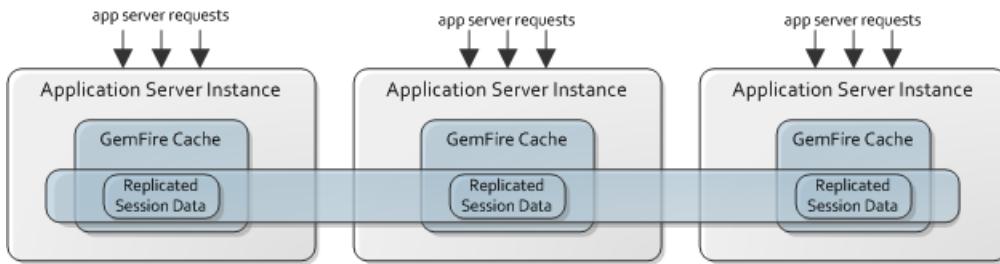


**Note:** The GemFire evaluation license allows you to create up to three GemFire processes. To use GemFire with additional processes, refer to [How GemFire Manages Licenses](#) on page 52.

### Setting Up the HTTP Module for Tomcat

To use the GemFire HTTP module with Tomcat application servers, you will need to modify Tomcat's `server.xml` and `context.xml` files. Configuration is slightly different depending on the topology you are setting up. Refer to [Common Topologies for HTTP Session Management](#) on page 761 for more information.

### Peer-to-Peer Setup



To run GemFire in a peer-to-peer configuration, add the following line to Tomcat's \${CATALINA\_HOME}/conf/server.xml within the <Server> tag:

```
<Listener className="com.gemstone.gemfire.modules.session.catalina.
    PeerToPeerCacheLifecycleListener"/>
```

Depending on the version of Tomcat you are using, add one of the following lines to \${CATALINA\_HOME}/conf/context.xml within the <Context> tag:

For Tomcat 6.0:

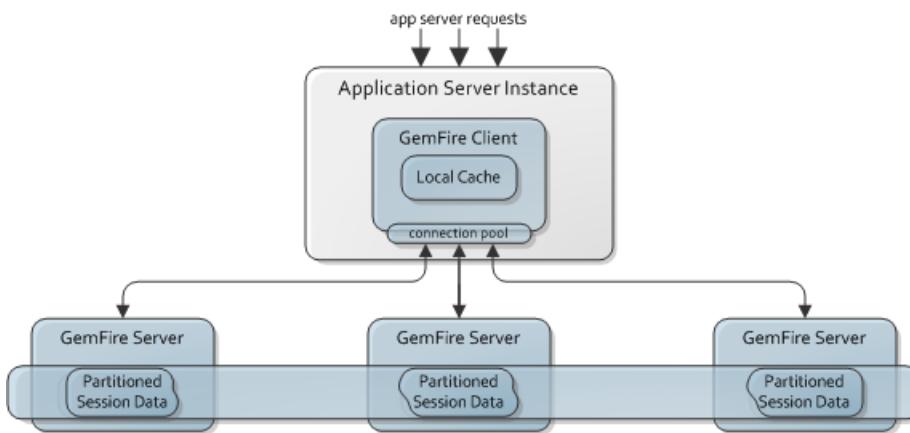
```
<Manager className="com.gemstone.gemfire.modules.session.catalina.
    Tomcat6DeltaSessionManager"/>
```

For Tomcat 7.0:

```
<Manager className="com.gemstone.gemfire.modules.session.catalina.
    Tomcat7DeltaSessionManager"/>
```

Without making any further configuration changes, peers attempt to locate each other using a default multicast channel (as defined by the mcast-port property found in [Changing GemFire Distributed System Properties](#) on page 793). Refer to [Peer-to-Peer Configuration Using Locators](#) on page 797 for information on creating and using Locators, which provide both discovery and load balancing services.

## Client/Server Setup



To run GemFire in a client/server configuration, the application server will operate as a GemFire client. To do this, add the following line to \${CATALINA\_HOME}/conf/server.xml within the <Server> tag:

```
<Listener className="com.gemstone.gemfire.modules.session.catalina.
    ClientServerCacheLifecycleListener"/>
```

Depending on the version of Tomcat you are using, add one of the following lines to \$CATALINA\_HOME\$/conf/context.xml within the <Context> tag:

For Tomcat 6.0:

```
<Manager className="com.gemstone.gemfire.modules.session.catalina.
    Tomcat6DeltaSessionManager" />
```

For Tomcat 7.0:

```
<Manager className="com.gemstone.gemfire.modules.session.catalina.
    Tomcat7DeltaSessionManager" />
```

Since the application server operates as a GemFire client in this configuration, you must manually launch a GemFire cache server. You can do this with the following script (which should be installed into the bin subdirectory):

In Unix:

```
prompt$ ./cacheserver.sh start
```

In Windows:

```
prompt> cacheserver.bat start
```

See [vFabric GemFire Servers](#) on page 741 for more information on using this script.



**Note:** If you plan to use the cacheserver script that comes with the standalone GemFire product, you will need to manually add the following items to the classpath: CATALINA\_HOME/lib/servlet-api.jar, CATALINA\_HOME/lib/catalina.jar, CATALINA\_HOME/bin/tomcat-juli.jar, CATALINA\_HOME/conf. Note that these items are automatically added to the classpath by the supplied cacheserver.sh (or cacheserver.bat) script that comes with the GemFire HTTP Session Management module.

Without making any further configuration changes, the client and server attempt to locate each other on the same system using a default port (40404). Though useful for testing purposes, you would not normally deploy a client/server configuration in this way. Refer to [Client/Server Configuration Using Locators](#) on page 798 for information on creating and using Locators, which provide both discovery and load balancing services.

## Starting the Application Server

Once you've updated the configuration, you are now ready to start your tc Server or Tomcat instance. Refer to your application server documentation for starting the application server. Once started, GemFire will automatically launch within the application server process.



**Note:** GemFire session state management provides its own clustering functionality. If you are using GemFire, you should NOT turn on Tomcat clustering as well.

## Verifying that GemFire Started

You can verify that GemFire has successfully started by inspecting the Tomcat log file. For example:

```
Nov 8, 2010 12:12:12 PM
com.gemstone.gemfire.modules.session.catalina.ClientServerCacheLifecycleListener
createOrRetrieveCache
INFO: Created GemFireCache[id = 2066231378; isClosing = false;
      created = Mon Nov 08 12:12:12 PDT 2010; server = false;
      copyOnRead = false; lockLease = 120; lockTimeout = 60]
```

Information is also logged within the GemFire log file, which by default is named "gemfire\_modules.log".

## Changing the Default GemFire Configuration in the Tomcat Module

By default, the Tomcat module will run GemFire automatically with pre-configured settings. You can change these GemFire settings.

Here are the default settings:

- GemFire peer-to-peer members are discovered using a default multicast channel.
- GemFire locators are not used for member discovery.
- The region name is set to `gemfire_modules_sessions`.
- The cache region is replicated for peer-to-peer configurations and partitioned (with redundancy turned on) for client/server configurations.
- GemFire clients have local caching turned on and when the local cache needs to evict data, it will evict least-recently-used (LRU) data first.



**Note:** On the application server side, the default inactive interval for session expiration is set to 30 minutes. To change this value, refer to [Session Expiration](#) on page 763.

However, you may want to change this default configuration. For example, you might want to change the region from replicated to partitioned, or use locators for discovery, or set up a multi-site configuration. This section describes how to change these configuration values.

## Changing GemFire Distributed System Properties

To edit GemFire system properties (such as how GemFire members locate each other), you must add properties to Tomcat's `server.xml` file. When adding properties, use the following syntax:

```
<Listener
  className="com.gemstone.gemfire.modules.session.catalina.xxxLifecycleListener"
  name="value"
  name="value"
  ...
/>
```

In the preceding code snippet, `xxxLifecycleListener` is either a `PeerToPeerCacheLifecycleListener` or a `ClientServerCacheLifecycleListener` depending on your configuration; `name` is the property name and `value` is the property value. For example:

```
<Listener className="com.gemstone.gemfire.modules.session.catalina.
  PeerToPeerCacheLifecycleListener"
  cache-xml-file="cache-peer.xml"
  mcast-port=0
  locators="hostname[10334]"
/>
```

This example specifies that the filename for GemFire's cache xml configuration is `cache-peer.xml` and that a locator can be found at "hostname : 10334". Note that when using a locator, you must turn off multicast discovery by setting the `mcast-port` property to 0. Locators are explained in more detail in [Common GemFire Configuration Changes in Tomcat](#) on page 782.

The list of configurable Tomcat's `server.xml` system properties include any of the properties that can be specified in GemFire's `gemfire.properties` file. The following list contains some of the more common parameters that can be configured.

Parameter	Description	Default
cache-xml-file	name of the cache configuration file	cache-peer.xml for peer-to-peer, cache-client.xml for client/server
locators (only for peer-to-peer config)	list of locators (host[port]) used by GemFire members to discover each other; if this value is empty, then multicast discovery will take place instead; when using a locator, you should turn off multicast discovery by setting mcast-port to 0 and you must manually start the locator using a command such as "gemfire start-locator"; refer to " <a href="#">Peer-to-Peer Configuration Using Locators</a> on page 797" for more information	empty string (i.e. use multicast instead)
log-file	name of the GemFire log file	gemfire_modules.log
mcast-port (only for peer-to-peer config)	port used by GemFire members to discover each other using multicast networking; when using locators in place of multicast, this value should be disabled by setting it to 0	10334
statistic-archive-file	name of the GemFire statistics file	gemfire_modules.gfs
statistic-sampling-enabled	whether GemFire statistics sampling is enabled	false

For more information on these properties, along with the full list of properties, refer to [gemfire.properties and gfsecurity.properties \(GemFire Properties\)](#) on page 851.

In addition to the standard GemFire system properties, the following cache-specific properties can also be configured with the `LifecycleListener`.

Parameter	Description	Default
criticalHeapPercentage	percentage of heap at which updates to the cache are refused	0 (disabled)
evictionHeapPercentage	percentage of heap at which session eviction begins	80.0
rebalance	whether a rebalance of the cache should be done when the application server instance is started	false

Although these properties are not part of the standard GemFire system properties, they apply to the entire JVM instance and are therefore also handled by the `LifecycleListener`. For more information about managing the heap, refer to [Heap Use and Management](#) on page 453.

## Changing Cache Configuration Properties

To edit GemFire cache properties (such as the name and the characteristics of the cache region), you must add these properties to Tomcat's `context.xml` file. When adding properties, unless otherwise specified, use the following syntax:

```
<Manager
    className="com.gemstone.gemfire.modules.session.catalina.Tomcat7DeltaSessionManager"
        name="value"
        name="value"
        ...
    />
```

In the preceding code snippet, name is the property name and value is the property value. For example:

```
<Manager className="com.gemstone.gemfire.modules.session.catalina.
    Tomcat7DeltaSessionManager"
    regionAttributesId="PARTITION_REDUNDANT"
    regionName="my_region"
/>
```

This example creates a partitioned region by the name of "my\_region".

The following parameters are the cache configuration parameters that can be added to Tomcat's `context.xml` file.

#### **CommitSessionValve**

Whether to wait until the end of the HTTP request to save all session attribute changes to the GemFire cache; if the configuration line is present in the application's `context.xml` file, then only one put will be performed into the cache for the session per HTTP request. If the configuration line is not included, then the session is saved each time the `setAttribute` or `removeAttribute` method is invoked. As a consequence, multiple puts are performed into the cache during a single session. This configuration setting is recommended for any applications that modify the session frequently during a single HTTP request.

Default: Set

To disable this configuration, remove or comment out the following line from Tomcat's `context.xml` file.

```
<Valve
    className="com.gemstone.gemfire.modules.session.catalina.CommitSessionValve"/>
```

#### **enableDebugListener**

Whether to enable a debug listener in the session region; if this parameter is set to true, info-level messages are logged to the GemFire log when sessions are created, updated, invalidated or expired

Default: false

The GemFire API equivalent to setting this parameter:

```
// Create factory
AttributesFactory factory = ...; <or> RegionFactory factory =
...
// Add cache listener
factory.addCacheListener(new DebugCacheListener());
```

#### **enableGatewayReplication**

Whether to enable a gateway; if this parameter is set to true, a `GatewayHub` must be configured; see [Multi-site Setup](#) on page 799 for more information

Default: false

The GemFire API equivalent to setting this parameter:

```
// Create factory
AttributesFactory factory = ...; <or> RegionFactory<> factory =
...
// Set enable gateway
factory.setEnableGateway(true);
```

#### **enableLocalCache**

Whether a local cache is enabled. If this parameter is set to true, the app server load balancer should be configured for sticky session mode

Default: `false` for peer-to-peer, `true` for client/server

The GemFire API equivalent to setting this parameter:

```
// For peer-to-peer members:  
Cache.createRegionFactory(REPLICATE_PROXY)  
// For client members:  
ClientCache.createClientRegionFactory(CACHING_PROXY_HEAP_LRU)
```

#### **regionAttributesId**

Specifies the region shortcut. For more information see [Region Shortcuts and Custom Named Region Attributes](#) on page 150; when using a partitioned region attribute, it is recommended that you use PARTITION\_REDUNDANT (rather than PARTITION) to ensure that the failure of a server does not result in lost session data

Default: `REPLICATE` for peer-to-peer, `PARTITION_REDUNDANT` for client/server

The GemFire API equivalent to setting this parameter:

```
// Creates a region factory for the specified region shortcut  
Cache.createRegionFactory(regionAttributesId);
```

#### **regionName**

Name of the region

Default: `gemfire_modules_sessions`

The GemFire API equivalent to setting this parameter:

```
// Creates a region with the specified name  
RegionFactory.create(regionName);
```

Refer to [Cache Management](#) on page 137 for more information about configuring the cache.

### **Common GemFire Configuration Changes in Tomcat**

#### **Using a Different Multicast Port**

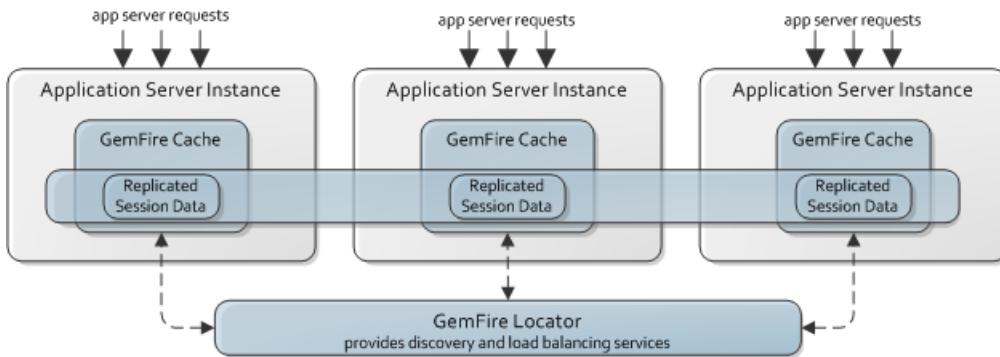
For a GemFire peer-to-peer member to communicate on a different multicast port than the default (10334), use the `mcast-port` property in Tomcat's `server.xml` file.

```
<Listener className="com.gemstone.gemfire.modules.session.catalina.  
          PeerToPeerCacheLifecycleListener"  
        locators="" mcast-port=10445  
      />
```

This example uses port 10445 as the multicast port.

#### **Peer-to-Peer Configuration Using Locators**

By default, GemFire peers discover each other using multicast communication on a known port (as specified by the `mcast-port` system parameter). Alternatively, you can configure member discovery using one or more dedicated locators. A locator provides both discovery and load balancing services.



To run GemFire with one or more locators instead of a multicast port, use the `locators` property in Tomcat's `server.xml` file.

```
<Listener className="com.gemstone.gemfire.modules.session.catalina.
    PeerToPeerCacheLifecycleListener"
    mcast-port=0 locators="hostname[10334]"
/>
```

This example uses a locator at `hostname` on port 10334.

You must be sure when you specify a locator that you specifically launch a locator correctly at this location. You can do this using the `gemfire` command-line tool found in the `bin` subdirectory of the HTTP module distribution.

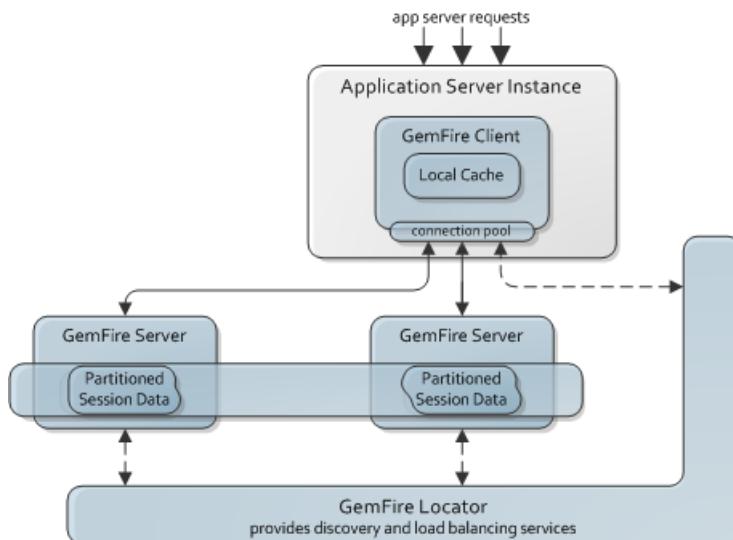
```
In Unix:
./gemfire.sh start-locator -port=10334
```

```
In Windows:
gemfire.bat start-locator -port=10334
```

This example starts a locator that listens on port 10334.

### **Client/Server Configuration Using Locators**

By default, GemFire clients and servers discover each other on a pre-defined port on the localhost. This is not typically the way you would deploy a client/server configuration. A preferred solution is to use one or more dedicated locators. A locator provides both discovery and load balancing services.



To run GemFire with one or more locators instead of a multicast port, add locator information to the `cache-client.xml` file in the Tomcat instance's `conf` subdirectory:

```
<pool name="sessions">
  <locator host="hostname" port="10334"/>
</pool>
```

This example tells the GemFire client (which is the process running the application server) to use a locator at `hostname` on port 10334.

You must now launch a locator correctly at this location. You can do this using the `gemfire` command-line tool found in the `bin` subdirectory of the HTTP module distribution..

```
./gemfire start-locator -port=10334
```

This example starts a locator that listens on port 10334.

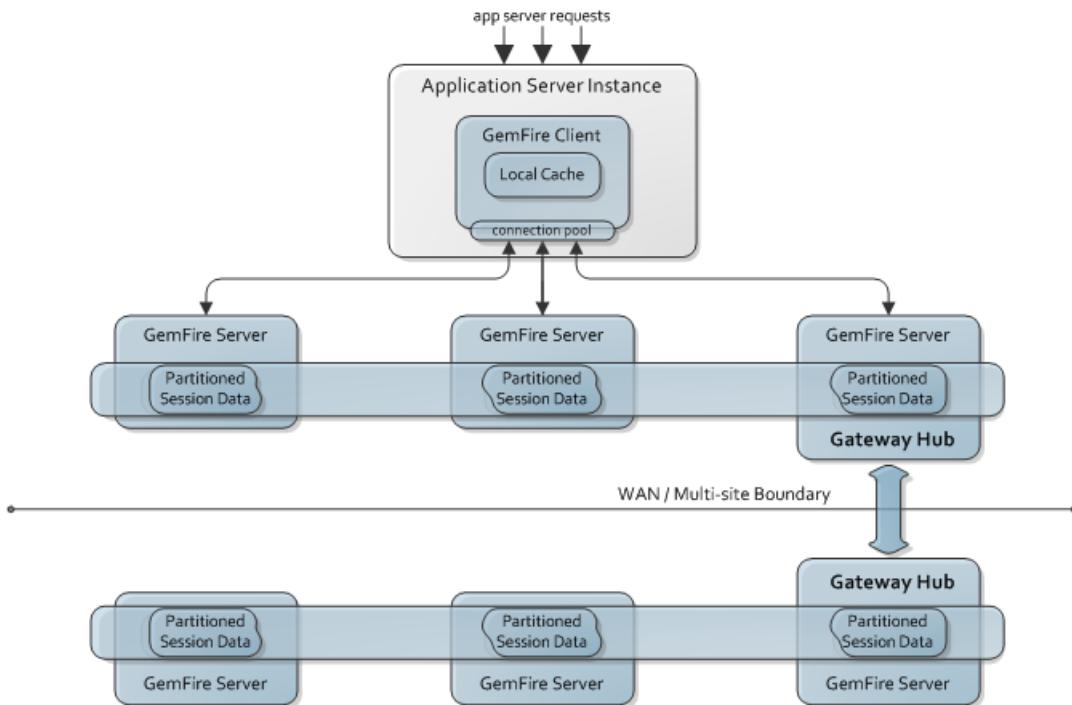
You also need to tell the GemFire server to use a locator when you launch the server. You can do this through a command-line argument when you start up the server with the script provided in the `bin` subdirectory wherever you installed the GemFire HTTP module:

```
In Unix:
prompt$ ./cacheserver.sh start locators=hostname[port]

In Windows:
prompt> cacheserver.bat start locators=hostname[port]
```

If you are running more than one locator, use a comma-separated list of locators.

## **Multi-site Setup**



For information on when to use this topology, refer to [Common Topologies for HTTP Session Management](#) on page 761. To enable gateway communication between sites, add one of the following line to Tomcat's `context.xml` file according to the version of Tomcat you are using.

For Tomcat 6:

```
<Manager className="com.gemstone.gemfire.modules.session.catalina.
    Tomcat6DeltaSessionManager"
    enableGatewayReplication=true
/>
```

For Tomcat 7:

```
<Manager className="com.gemstone.gemfire.modules.session.catalina.
    Tomcat7DeltaSessionManager"
    enableGatewayReplication=true
/>
```

You will now need to modify the GemFire configuration file (either `cache-peer.xml` for a peer-to-peer configuration or `cache-server.xml` for a client/server configuration) in the `conf` subdirectory of the HTTP module instance that will operate as the gateway hub. For example, if you were setting up a gateway between a site named "SITE1" and a site named "SITE2", this is how you would set up the information for site 1:

```
<gateway-hub id="SITE1" port="22222" socket-buffer-size="256000">
    <gateway id="SITE2_CLUSTERA" socket-buffer-size="256000">
        <gateway-endpoint id="SITE2_CLUSTERA_server1" host="site2_hostname"
            port="22222"/>
    </gateway>
</gateway-hub>
```

And at site 2, you would need to set up another gateway hub that communicates with site 1. Notice that the endpoint for site 1 references information about site 2 while the endpoint for site 2 references information about site 1.

```
<gateway-hub id="SITE2" port="22222" socket-buffer-size="256000">
  <gateway id="SITE1_CLUSTERA" socket-buffer-size="256000">
    <gateway-endpoint id="SITE1_CLUSTERA_server1" host="site1_hostname"
      port="22222" />
  </gateway>
</gateway-hub>
```



**Note:** To successfully run a configuration using a multi-site topology, you must start at least one server and one client in **each** site before creating or accessing your session. If a GemFire client is not started in one site, the region data containing the session information will not get initialized. This means that a web server must be started on **each** site before creating or accessing session data.

See [Multi-site \(WAN\) Configuration](#) on page 189 for more information.

## HTTP Session Management Module for AppServers

You implement session caching with the HTTP Session Management Module for AppServers with a special filter, defined in the `web.xml`, which is configured to intercept and wrap all requests.

You can use this HTTP module with a variety of application servers. Wrapping each request allows the interception of `getSession()` calls to be handled by GemFire instead of the native container. This approach is a generic solution, which is supported by any container that implements the Servlet 2.4 specification. (See [HTTP Session Management Quick Start](#) on page 757 for a list of tested application servers.)

### Installing the HTTP Module for AppServers

Typically, all necessary jar files can be bundled within the WAR or EAR file you want to deploy. However, in some situations the core GemFire and module JARs may need to be shared across web application deployments.

The following instructions describe how to install the session management module in your application server.

1. Download and unpack the [HTTP Session Management for AppServers Module](#) into a directory on the application server.
2. From the directory where you unpacked the files, copy the following files from `lib` to an appropriate location on your application server (see table below) in order to make them available to all applications.

- `gemfire-modules-<version>.jar`
- `gemfire-modules-session-<version>.jar`
- `slf4j-api-<version>.jar`
- `slf4j-jdk14-<version>.jar *`

From the `<gemfire_product_dir>/lib` directory (where `gemfire_product_dir` corresponds to the location where you installed GemFire), copy the following files to an appropriate location on your application server (see table below) in order to make them available to all applications.

- `gemfire.jar`
- `antlr.jar`
- `spring-core-3.1.1.RELEASE.jar`
- `spring-shell-1.0.0.RELEASE.jar`

Application Server	Version	Location
WebLogic *	11g (10.3.x)	<code>&lt;domain&gt;/lib</code>

Application Server	Version	Location
WebSphere	7	<websphere install dir>/lib/ext
WebSphere	8	Consult application server's documentation
JBoss AS	5	<jboss install dir>/lib
JBoss AS	6	<jboss install dir>/common/lib
JBoss AS	7	Consult application server's documentation.

\* For WebLogic, substitute `gemfire-modules-slf4j-weblogic-<version>.jar` for `slf4j-jdk14-<version>.jar`.



**Note:** This table lists all application server versions that have been tested with the GemFire HTTP Session Management modules. However, the generic HTTP Session Management Module for AppServers is implemented as a servlet filter and should work on any application server platform that supports the Java Servlet 2.4 specification.



**Note:** The GemFire evaluation license allows you to create up to three GemFire processes. To use GemFire with additional processes, refer to [How GemFire Manages Licenses](#) on page 52.

## Setting Up the HTTP Module for AppServers

To use the module, you need to modify your application's `web.xml` files. Configuration is slightly different depending on the topology you are setting up.

- [Using the modify\\_war Script](#) on page 787
- [Manual Configuration](#) on page 789
- [Peer-to-Peer Setup](#) on page 790
- [Client/Server Setup](#) on page 791
- [Starting the Application Server](#) on page 792
- [Verifying that GemFire Started](#) on page 792
- [Configuring Non-Sticky Session Replication](#) on page 792

Refer to [Common Topologies for HTTP Session Management](#) on page 761 for more information. Modifying the war file is typically done with the `modify_war` script, but could also be performed manually.

## Using the `modify_war` Script

The script has the following options:

```
USAGE: modify_war <args>
WHERE <args>:
```

```
-e <jar>          Assumes the input file is an .ear file and will add the
                  given jar as a shared, application library. The file
                  will
                  be added in a /lib directory (by default) and any
                  embedded
                  .war files will have a corresponding Class-Path entry
                  added
                  to their MANIFEST.MF file. The option can be given
                  multiple times.
```

```

-h
    Displays this help message.

-j <jar>
    Additional library to add to the input file. Can be
given
multiple times.

-J <jvm opt>
    JVM argument to pass to sub-commands. Typically this
might
be to define proxy values. For example
-J"-Dhttp.proxyHost=my-proxy"

-l <lib>
    Library directory where new jars will be placed inside
war.
Defaults to WEB-INF/lib.

-m <lib>
    Library directory where new jars will be placed inside
ear.
Defaults to /lib.

-o <file>
    The output file.

-p <param=value>
    Specific parameter for inclusion into the session filter
definition as a regular init-param. Can be given multiple
times.

-r
    Test run which only outputs an updated web.xml file.

-t <cache-type>
    Type of cache. Must be one of 'peer-to-peer' or
'client-server'. Default is peer-to-peer.

-v
    Verbose output

-w <war/ear file>
    The input file - either a WAR or EAR. The following
actions
will be performed depending on the type of file:
WARs will have a <filter> element added to the web.xml
and
will have the appropriate jars copied to WEB-INF/lib.
If the file is an EAR, then the appropriate jars will
be
copied to lib, within the ear and each embedded war
files'
manifest will have a Class-Path entry added (if one
does
not already exist).
ears
An appropriate slf4j binding jar must be included for
or wars using -e or -j respectively. The following jars

```

```

are
    provided:
        slf4j-jdk14
        slf4j-log4j12
        gemfire-modules-slf4j-weblogic

-x
Do not create a self-contained war/ear file by copying
all
necessary jars into the file. When this option is used,
additional jars will need to be made available to the
container:
    gemfire.jar
    gemfire-modules.jar
    gemfire-modules-session.jar
    slf4j-api.jar
    slf4j-jdk14.jar (not for WebLogic)
    gemfire-modules-slf4j-weblogic.jar (WebLogic only)

This option still modifies any web.xml files.

```

## Manual Configuration

To modify your war or ear file manually, make the following updates:

- **web.xml** needs a filter and listener added as follows. If you have your own filters, the GemFire Module filter **must** be the first one.

```

<filter>
    <filter-name>gemfire-session-filter</filter-name>
    <filter-class>
        com.gemstone.gemfire.modules.session.filter.SessionCachingFilter
    </filter-class>
    <init-param>
        <param-name>cache-type</param-name>
        <param-value>peer-to-peer</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>gemfire-session-filter</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>ERROR</dispatcher>
</filter-mapping>
<listener>

<listener-class>com.gemstone.gemfire.modules.session.filter.SessionListener</listener-class>
</listener>

```

- Add **gemfire-modules-session-external.jar** to the WEB-INF/lib directory of the war.

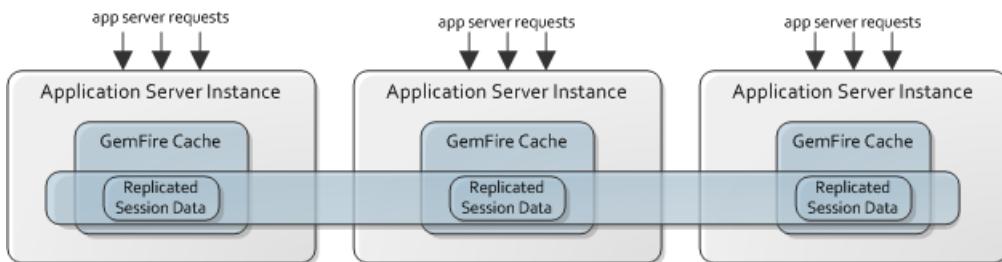
If you are deploying an ear file and wish to bundle all dependent jars within the ear then the following steps are also required:

- Copy the dependent files to the lib directory of the ear.
  - **gemfire.jar**
  - **gemfire-modules.jar**

- gemfire-modules-session.jar
- slf4j-api.jar
- slf4j-jdk14.jar (not necessary for WebLogic)
- gemfire-modules-slf4j-weblogic.jar (only for WebLogic)
- Modify each embedded war file's manifest by adding a Class-Path entry which references the shared jars added above. For example:

```
Manifest-Version: 1.0
Built-By: joe
Build-Jdk: 1.6.0_22
Created-By: Apache Maven
Archiver-Version: Plexus Archiver
Class-Path: lib/gemfire-6.6.3.jar
lib/gemfire-modules-2.1.1.1.jar
lib/gemfire-modules-session-2.1.1.1.jar
lib/slf4j-api-1.5.8.jar
lib/slf4j-jdk14-1.5.8.jar
```

## Peer-to-Peer Setup



To run GemFire in a peer-to-peer configuration, use the **-t peer-to-peer** option to the `modify_war` script. This adds the following to `web.xml`:

```
<filter>
    <filter-name>gemfire-session-filter</filter-name>
    <filter-class>
        com.gemstone.gemfire.modules.session.filter.SessionCachingFilter
    </filter-class>
    <init-param>
        <param-name>cache-type</param-name>
        <param-value>peer-to-peer</param-value>
    </init-param>
</filter>
```

If you do not make any further configuration changes, peers attempt to locate each other using a default multicast channel (as defined by the `mcast-port` property found in [Using a Different Multicast Port](#) on page 796). Refer to [Peer-to-Peer Configuration Using Locators](#) on page 797 for information on creating and using locators, which provide both discovery and load balancing services.

GemFire properties can be set with filter params prefixed with `gemfire.property`. Thus, to use a specific multicast port you would use the following option: `-p gemfire.property.mcast-port=12345 -t peer-to-peer`. This would create the xml below:

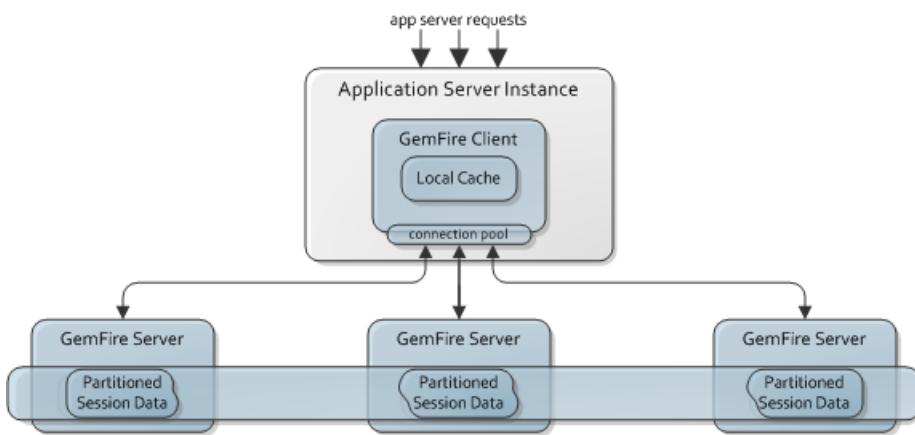
```
<filter>
    <filter-name>gemfire-session-filter</filter-name>
    <filter-class>
        com.gemstone.gemfire.modules.session.filter.SessionCachingFilter
    </filter-class>
    <init-param>
        <param-name>cache-type</param-name>
        <param-value>peer-to-peer</param-value>
    </init-param>
    <param>
        <param-name>gemfire.property.mcast-port</param-name>
        <param-value>12345</param-value>
    </param>
</filter>
```

```

</filter-class>
<init-param>
    <param-name>cache-type</param-name>
    <param-value>peer-to-peer</param-value>
</init-param>
<init-param>
    <param-name>gemfire.property.mcast-port</param-name>
    <param-value>12345</param-value>
</init-param>
</filter>

```

## Client/Server Setup



To run GemFire in a client/server configuration, you make the application server operate as a GemFire client. Use the `-t client-server` option to the `modify_war` script. This adds the following filter to application server's `web.xml` file:

```

<filter>
    <filter-name>gemfire-session-filter</filter-name>
    <filter-class>
        com.gemstone.gemfire.modules.session.filter.SessionCachingFilter
    </filter-class>
    <init-param>
        <param-name>cache-type</param-name>
        <param-value>client-server</param-value>
    </init-param>
</filter>

```

Because the application server operates as a GemFire client in this configuration, you must manually launch a GemFire cache server. You can do this with the following script (which should be installed into the bin subdirectory):

```

In Unix:
prompt$ ./cacheserver.sh start

In Windows:
prompt> cacheserver.bat start

```

See [vFabric GemFire Servers](#) on page 741 for more information on using this script.

If you plan to use the deprecated `cacheserver` script that comes with the standalone GemFire product, you will need to manually add the following items to the classpath. Note that these items are automatically added to the

classpath by the supplied cacheserver.sh (or cacheserver.bat) script that comes with the GemFire HTTP Session Management Module for AppServers.

- lib/gemfire.jar
- lib/gemfire-modules.jar
- lib/gemfire-modules-session.jar
- lib/slf4j-api.jar
- lib/slf4j-jdk14.jar

Without making any further configuration changes, the client and server attempt to locate each other on the same system using a default port (40404). Though useful for testing purposes, you would not normally deploy a client/server configuration in this way. Refer to [Client/Server Configuration Using Locators](#) on page 798 for information on creating and using Locators, which provide both discovery and load balancing services.

### Starting the Application Server

After you update the configuration, you are now ready to start your application server instance. Refer to your application server documentation for starting the application server. Once the application server is started, GemFire will automatically launch within the application server process.

### Verifying That GemFire Started

You can verify that GemFire has successfully started by inspecting the application server log file. For example:

```
####<Feb 22, 2011 10:34:34 AM PST> <Info> ... <BEA-000000>
<Creating distributed system from:
 {log-file=user/weblogic/domain1/servers/myserver/
  logs/gemfire_modules.2011-02-22.log,
  cache-xml-file=cache.xml}>
####<Feb 22, 2011 10:34:39 AM PST> <Info> ... <BEA-000000>
<Created GemFireCache[id = 28354945; isClosing = false;
 created = Tue Feb 22 10:34:39 PST 2011; server = false;
 copyOnRead = false; lockLease = 120; lockTimeout = 60]>
```

Information is also logged within the GemFire log file, which by default is named "gemfire\_modules.<date>.log".

### Configuring Non-Sticky Session Replication

Some situations require sessions to be 'non-sticky', which means that client requests can be directed to any server in a cluster of application servers instead of the same server each time. To achieve this, you must configure your deployment as described for the following topologies.

#### Peer-to-Peer

No additional configuration is required.

#### Client-Server

For client-server topologies, the local client cache must be empty. Ensure that the filter property `gemfire.cache.enable_local_cache=false` is set. This effectively sets the local client cache to be a **PROXY** cache.



**Note:** Non-sticky sessions affect performance because sessions need to be re-created every time a request hits a different server. This may not be noticeable when the session attributes are small, but may become more evident as the session attributes increase in size and/or number.

## Changing the Default GemFire Configuration in the AppServers Module

By default, the AppServers module will run GemFire automatically with preconfigured settings. You can change these GemFire settings.

Here are the default settings:

- GemFire peer-to-peer members are discovered using a default multicast channel.
- GemFire locators are not used for member discovery.
- The region name is set to `gemfire_modules_sessions`.
- The cache region is replicated for peer-to-peer configurations and partitioned (with redundancy turned on) for client/server configurations.
- GemFire clients have local caching turned on and when the local cache needs to evict data, it will evict least-recently-used (LRU) data first.



**Note:** On the application server side, the default inactive interval for session expiration is set to 30 minutes. To change this value, refer to [Session Expiration](#) on page 763.

However, you may want to change this default configuration. For example, you might want to change the region from replicated to partitioned, or use locators for discovery, or set up a multi-site configuration. This section describes how to change these configuration values.



**Note:** You cannot override region attributes on the cache server when using the HTTP Session Management Module. You must place all region attribute definitions in the region attributes template that you customize in your application server. See [Overriding Region Attributes](#) on page 797 for more information.

## Changing GemFire Distributed System Properties

To edit GemFire system properties (such as how GemFire members locate each other), you must add properties to GemFire Session Filter definition in the application's web.xml file. As mentioned above, this can be done by using the `-p` option to the `modify_war` script. All GemFire system properties should be prefix with the string `gemfire.property`. For example:

- `-p gemfire.property.mcast-port=0`
- `-p gemfire.property.locators=hostname[10334]`
- `-p gemfire.property.cache-xml-file=/u01/weblogic/conf/cache.xml`

```
<filter>
    <filter-name>gemfire-session-filter</filter-name>
    <filter-class>
        com.gemstone.gemfire.modules.session.filter.SessionCachingFilter
    </filter-class>
    <init-param>
        <param-name>cache-type</param-name>
        <param-value>client-server</param-value>
    </init-param>
    <init-param>
        <param-name>gemfire.property.mcast-port</param-name>
        <param-value>0</param-value>
    </init-param>
    <init-param>
        <param-name>gemfire.property.locators</param-name>
        <param-value>hostname[10334]</param-value>
    </init-param>
    <init-param>
        <param-name>gemfire.property.cache-xml-file</param-name>
        <param-value>/u01/weblogic/conf/cache.xml</param-value>
    </init-param>
```

```
</init-param>
</filter>
```

This example specifies that the filename for GemFire's cache XML configuration is cache-peer.xml and that a locator can be found at "hostname:10334". Note that when using a locator, you must turn off multicast discovery by setting the mcast-port property to 0.

The list of configurable **server.xml** system properties include any of the properties that can be specified in GemFire's `gemfire.properties` file. The following list contains some of the more common parameters that can be configured.

Parameter	Description	Default
cache-xml-file	Name of the cache configuration file.	cache-peer.xml for peer-to-peer, cache-client.xml for client/server
locators (only for peer-to-peer config)	List of locators (host[port]) used by GemFire members to discover each other; if this value is empty, then multicast discovery will take place instead; when using a locator, you should turn off multicast discovery by setting <code>mcast-port</code> to 0 and you must manually start the locator using a command such as " <code>gemfire start-locator</code> "; refer to " <a href="#">Peer-to-Peer Configuration Using Locators</a> on page 797" for more information.	Empty string (i.e. use multicast instead)
log-file	Name of the GemFire log file.	gemfire_modules.log
mcast-port (only for peer-to-peer config)	Port used by GemFire members to discover each other using multicast networking; when using locators in place of multicast, this value should be disabled by setting it to 0.	10334
statistic-archive-file	Name of the GemFire statistics file.	gemfire_modules.gfs
statistic-sampling-enabled	Whether GemFire statistics sampling is enabled.	false

For more information on these properties, along with the full list of properties, refer to [gemfire.properties and gfsecurity.properties \(GemFire Properties\)](#) on page 851.

In addition to the standard GemFire system properties, the following cache-specific properties can also be configured.

Parameter	Description	Default
criticalHeapPercentage	Percentage of heap at which updates to the cache are refused.	0 (disabled)
evictionHeapPercentage	Percentage of heap at which session eviction begins.	80.0
rebalance	Whether a rebalance of the cache should be done when the application server instance is started.	false

Although these properties are not part of the standard GemFire system properties, they apply to the entire JVM instance. For more information about managing the heap, refer to [Heap Use and Management](#) on page 453.



**Note:** It is important to note that the GemFire Distributed System is a singleton within the entire application server JVM. As such it is important to ensure that different web applications, within the same container, set (or expect) the same cache configuration. When the application server starts, the first web application to start that uses GemFire Session Caching will determine the overall configuration of the distributed system since it will trigger the creation of the distributed system.

## Changing Cache Configuration Properties

To edit GemFire cache properties (such as the name and the characteristics of the cache region), you must configure these using a filter initialization parameter prefix of **gemfire.cache** with the `modify_war` script. For example:

**-p gemfire.cache.region\_name=custom\_sessions**

```
<filter>
  <filter-name>gemfire-session-filter</filter-name>
  <filter-class>
    com.gemstone.gemfire.modules.session.filter.SessionCachingFilter
  </filter-class>
  <init-param>
    <param-name>cache-type</param-name>
    <param-value>peer-to-peer</param-value>
  </init-param>
  <init-param>
    <param-name>gemfire.cache.region_name</param-name>
    <param-value>custom_sessions</param-value>
  </init-param>
</filter>
```

The following parameters are the cache configuration parameters that can be added to the filter definition as initialization parameters.

### enable\_debug\_listener

Whether to enable a debug listener in the session region; if this parameter is set to true, info-level messages are logged to the GemFire log when sessions are created, updated, invalidated or expired.

Default: `false`

The GemFire API equivalent to setting this parameter:

```
// Create factory
AttributesFactory factory = ....;
<or> RegionFactory factory = ....;
// Add cache listener
factory.addCacheListener(new DebugCacheListener());
```

### enable\_gateway\_replication

Whether to enable a gateway; if this parameter is set to true, a `GatewayHub` must be configured; see [Multi-site Setup](#) on page 799 for more information.

Default: `false`

The GemFire API equivalent to setting this parameter:

```
// Create factory
AttributesFactory factory = ....;
<or> RegionFactory factory = ....;
// Set enable gateway
factory.setEnableGateway(true);
```

### enable\_local\_cache

Whether a local cache is enabled; if this parameter is set to true, the app server load balancer should be configured for sticky session mode.

Default: `false` for peer-to-peer, `true` for client/server

The GemFire API equivalent to setting this parameter:

```
// For peer-to-peer members:  
Cache.createRegionFactory(REPLICATE_PROXY)  
// For client members:  
ClientCache.createClientRegionFactory(CACHING_PROXY_HEAP_LRU)
```

#### **region\_attributes\_id**

Specifies the region shortcut; for more information refer to [Region Shortcuts and Custom Named Region Attributes](#) on page 150; when using a partitioned region attribute, it is recommended that you use PARTITION\_REDUNDANT (rather than PARTITION) to ensure that the failure of a server does not result in lost session data.

Default: REPLICATE for peer-to-peer, PARTITION\_REDUNDANT for client/server

The GemFire API equivalent to setting this parameter:

```
// Creates a region factory for the specified region shortcut  
Cache.createRegionFactory(regionAttributesId);
```

#### **region\_name**

Name of the region.

Default: gemfire\_modules\_sessions

The GemFire API equivalent to setting this parameter:

```
// Creates a region with the specified name  
RegionFactory.create(regionName);
```

#### **session\_delta\_policy**

Replication policy for session attributes.

Default: delta\_queued

Delta replication can be configured to occur immediately when HttpSession.setAttribute() is called (delta\_immediate) or when the HTTP request has completed processing (delta\_queued). If the latter mode is configured, all attribute updates for a particular request are 'batched' and multiple updates to the same attribute are collapsed. Depending on the number of attributes updates within a given request, delta\_queued may provide a significant performance gain. For complete session attribute integrity across the cache, delta\_immediate is recommended. Note that this option is specific to this module and there is no equivalent GemFire API to enable it.

Refer to [Cache Management](#) on page 137 for more information about configuring the cache.

### **Common GemFire Configuration Changes for AppServers**

#### **Using a Different Multicast Port**

For a GemFire peer-to-peer member to communicate on a different multicast port than the default (10334), use the mcast-port property in the web.xml file either by manual configuration or by using -p gemfire.property.mcast-port=10445 option to the modify\_war script.

```
<filter>  
  <filter-name>gemfire-session-filter</filter-name>  
  <filter-class>  
    com.gemstone.gemfire.modules.session.filter.SessionCachingFilter  
  </filter-class>  
  <init-param>  
    <param-name>cache-type</param-name>  
    <param-value>peer-to-peer</param-value>  
  </init-param>
```

```

<init-param>
    <param-name>gemfire.property.mcast-port</param-name>
    <param-value>10445</param-value>
</init-param>
</filter>

```

This example uses port 10445 as the multicast port.

### Overriding Region Attributes

When using the HTTP Session Management Module, you cannot override region attributes directly on the cache server. You must place all region attribute definitions in the region attributes template that you customize within the application server. For example, to specify a different name for the region's disk store, you could add the new disk-store-name specification to the region attributes template and then reference the template on the cache server.

```

<region-attributes id="MY_SESSIONS"
refid="PARTITION_REDUNDANT_PERSISTENT_OVERFLOW"
disk-store-name="mystore">
...
</region-attributes>

```

Then on the cache server side, reference the modified region attributes template to allow the region to use the disk-store-name attribute:

```
<region name="gemfire_modules_sessions" refid="MY_SESSIONS" />
```

Next, you must specify the region attributes ID as a value for the `region_attributes_id` parameter in `web.xml`. For example, if you want to enable the `region-attributes` in the above example for a specific Web application, you would configure the Web application's `web.xml` in the following manner:

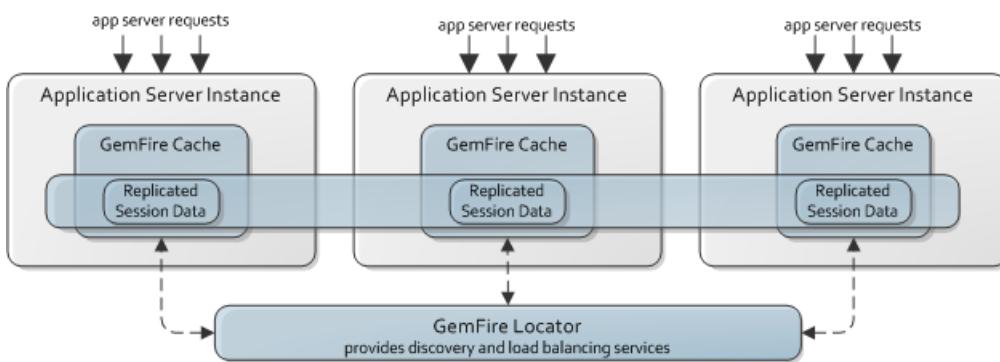
```

<filter>
...
<init-param>
    <param-name>gemfire.cache.region_attributes_id</param-name>
    <param-value>MY_SESSIONS</param-value>
</init-param>
...
</filter>

```

### Peer-to-Peer Configuration Using Locators

By default, GemFire peers discover each other using multicast communication on a known port (as specified by the `mcast-port` system parameter). Alternatively, you can configure member discovery using one or more dedicated locators. A locator provides both discovery and load balancing services.



To run GemFire with one or more locators instead of a multicast port, use the locators property in the web.xml file or the **-p gemfire.property.locators=hostname[10334]** option to the modify\_war script:

```
<filter>
    <filter-name>gemfire-session-filter</filter-name>
    <filter-class>
        com.gemstone.gemfire.modules.session.filter.SessionCachingFilter
    </filter-class>
    <init-param>
        <param-name>cache-type</param-name>
        <param-value>peer-to-peer</param-value>
    </init-param>
    <init-param>
        <param-name>gemfire.property.locators</param-name>
        <param-value>hostname[10334]</param-value>
    </init-param>
</filter>
```

This example uses a locator at hostname on port 10334.

You must be sure when you specify a locator that you specifically launch a locator correctly at this location. You can do this using the `gemfire` command-line tool found in the `bin` subdirectory of the HTTP module distribution.

In Unix:

```
./gemfire.sh start-locator -port=10334
```

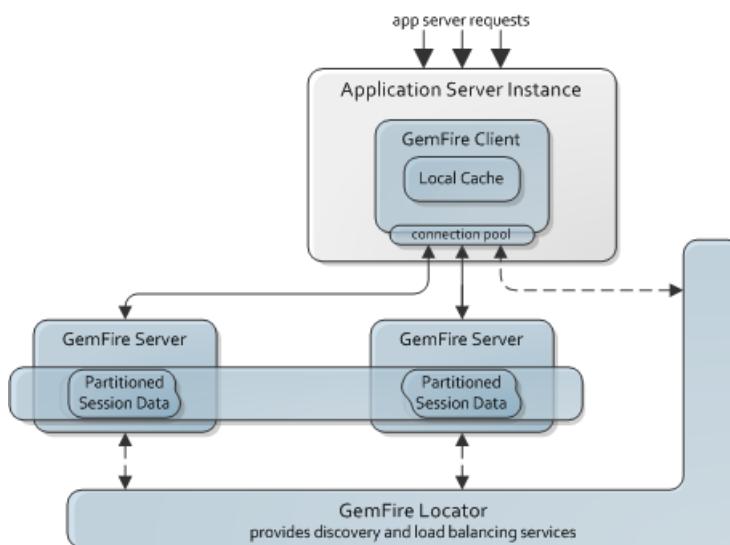
In Windows:

```
gemfire.bat start-locator -port=10334
```

This example starts a locator that listens on port 10334.

## Client/Server Configuration Using Locators

By default, GemFire clients and servers discover each other on a pre-defined port on the localhost. This is not typically the way you would deploy a client/server configuration. A preferred solution is to use one or more dedicated locators. A locator provides both discovery and load balancing services.



To run GemFire with one or more locators instead of a multicast port, add locator information to the `cache-client.xml` file in the AppServer module instance's `conf` subdirectory:

```
<pool name="sessions">
  <locator host="hostname" port="10334" />
</pool>
```

This example tells the GemFire client (which is the process running the application server) to use a locator at `hostname` on port 10334.

You must now launch a locator correctly at this location. You can do this using the `gemfire` command-line tool found in the `bin` subdirectory of the HTTP module distribution.

```
./gemfire start-locator -port=10334
```

This example starts a locator that listens on port 10334.

You also need to tell the GemFire server to use a locator when you launch the server. You can do this through a command-line argument when you start up the server with the script provided in the `bin` subdirectory wherever you installed the GemFire HTTP module:

In Unix:

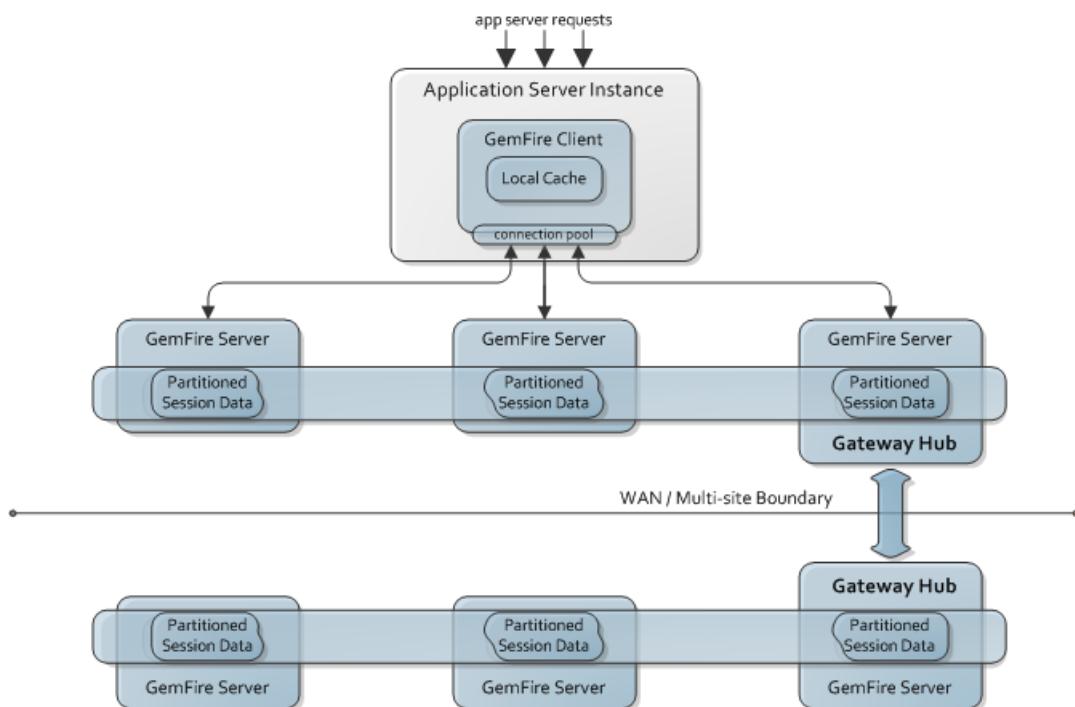
```
prompt$ ./cacheserver.sh start locators=hostname[port]
```

In Windows:

```
prompt> cacheserver.bat start locators=hostname[port]
```

If you are running more than one locator, use a comma-separated list of locators.

## Multisite Setup



For information on when to use this topology, refer to [Common Topologies for HTTP Session Management](#) on page 761. To enable gateway communication between sites, add the following line to the web application's web.xml file or use the **-p gemfire.cache.enable\_gateway\_replication=true** option of the `modify_war` script:

```
<filter>
    <filter-name>gemfire-session-filter</filter-name>
    <filter-class>
        com.gemstone.gemfire.modules.session.filter.SessionCachingFilter
    </filter-class>
    <init-param>
        <param-name>cache-type</param-name>
        <param-value>peer-to-peer</param-value>
    </init-param>
    <init-param>
        <param-name>gemfire.cache.enable_gateway_replication</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
```

You will now need to modify the GemFire configuration file (either `cache-peer.xml` for a peer-to-peer configuration or `cache-server.xml` for a client/server configuration) in the `conf` subdirectory of the HTTP module instance that will operate as the gateway hub. For example, if you were setting up a gateway between a site named "SITE1" and a site named "SITE2", this is how you would set up the information for site 1:

```
<gateway-hub id="SITE1" port="22222" socket-buffer-size="256000">
    <gateway id="SITE2_CLUSTERA" socket-buffer-size="256000">
        <gateway-endpoint id="SITE2_CLUSTERA_server1" host="site2_hostname"
            port="22222" />
    </gateway>
</gateway-hub>
```

And at site 2, you would need to set up another gateway hub that communicates with site 1. Notice that the endpoint for site 1 references information about site 2 while the endpoint for site 2 references information about site 1.

```
<gateway-hub id="SITE2" port="22222" socket-buffer-size="256000">
    <gateway id="SITE1_CLUSTERA" socket-buffer-size="256000">
        <gateway-endpoint id="SITE1_CLUSTERA_server1" host="site1_hostname"
            port="22222" />
    </gateway>
</gateway-hub>
```



**Note:** To successfully run a configuration using a multi-site topology, you must start at least one server and one client in **each** site before creating or accessing your session. If a GemFire client is not started in one site, the region data containing the session information will not get initialized. This means that a web server must be started on **each** site before creating or accessing session data.

See [Multi-site \(WAN\) Configuration](#) on page 189 for more information.

# Chapter 58

---

## Hibernate Cache Module

For additional information about Hibernate, see the [Hibernate](#) web site.

### Why Use Gemfire with Hibernate?

Using vFabric GemFire with Hibernate improves performance and scalability, gives you flexible caching options, and more.

- **General**

- GemFire's L2 cache solution improves Hibernate performance
- Reduces traffic to your database server
- Simple to configure: requires just a few changes to `hibernate.cfg.xml`
- Flexible caching options, supporting several local and distributed configurations
- Access to many of the GemFire features

- **Performance**

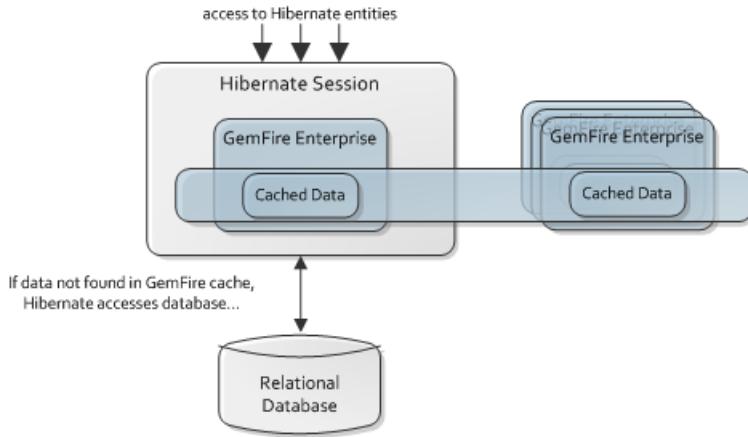
- Mesh-like architecture provides single hop access to any entity
- Designed to minimize serialization costs

- **Scalability**

- Data can be partitioned across the entire cluster
- Smart heap utilization / smart eviction algorithm
- Supports all Hibernate concurrency strategies (read only, read/write, non-strict read/write, transactional)
- Ensures data consistency without expensive distributed locks
- Overflow to disk on each system member (i.e. shared nothing persistence)
- Supports tiered caching

### How GemFire Works with Hibernate

After a few configuration changes, vFabric GemFire will cache Hibernate data. Depending on your usage model, you can replicate data across multiple peers, partition data across multiple servers, or manage your data in many other customizable ways.



## Installing the Hibernate Cache Module

The Hibernate module is included in the vFabric GemFire installation package.

1. If you have not done so, download and install [Hibernate](#). See [Hibernate Module Supported Configurations](#) on page 13 for currently supported versions of Hibernate.
2. If you have not done so, download and install vFabric GemFire 7.0 from the [vFabric GemFire download page](#) as described in [Installing vFabric GemFireInstall or update vFabric GemFire, and configure your environment to run it](#).

The Hibernate module is included in the vFabric GemFire 7.0 installation package. After you install vFabric GemFire, you can find the module in the following location:

```
<product_dir>/tools/Modules/vFabric_GemFire_Modules-7.0-Hibernate.zip
```

where <product\_dir> is the location where you installed GemFire.

3. Make sure that <product\_dir>/lib/gemfire.jar is part of the CLASSPATH when you run Hibernate. Alternatively, you can place the gemfire.jar in a location that is accessible to your Hibernate application.



**Note:** The GemFire evaluation license allows you to create up to three GemFire processes. To use GemFire with additional processes, see [Understanding vFabric GemFire Licenses](#) on page 45.

## Setting Up the GemFire Hibernate Cache Module

You need to edit Hibernate's hibernate.cfg.xml file in order to use the vFabric GemFire module.

### 1. Turn on L2 Cache

In the hibernate.cfg.xml file, turn on the L2 cache:

```
<property name="hibernate.cache.use_second_level_cache">true</property>
```

### 2. Set Region Factory or Cache Provider

Associate the region factory class with GemFireRegionFactory:

```
<property name="hibernate.cache.region.factory_class">
  com.gemstone.gemfire.modules.hibernate.GemFireRegionFactory
</property>
```

### 3. Determine Cache Usage Mode

Determine the cache usage mode for the entities in each region. There are four types of usage modes:

Mode	Description
Read Only	This mode is used when you do not plan on modifying the data already stored in your persistent storage.
Read/Write	This mode is used when you plan to both read from and write to your data.
Non-strict Read/Write	This mode is a special read/write mode that has faster write performance; however, only use this mode if no more than one client will update content at a time.
Transactional	This mode allows for transaction-based data access.

#### 4. Set Cache Usage Mode

The usage mode can either be set using the hibernate-mapping file or through Java annotations.

- To set the mode with the hibernate-mapping file, refer to this example:

```
<hibernate-mapping package="PACKAGE">
    <class name="ENTITY_NAME" ...>
        <cache usage="read-write|nonstrict-read-write|read-only"/>
        ...
    </class>
</hibernate-mapping>
```

In this example, PACKAGE is the name of the entity package and ENTITY\_NAME is the name of your entity. The cache usage values correspond to the cache usage modes, described above. Refer to the [Hibernate documentation](#) for further information.

- To set the mode using annotations, your class definition should look something like this:

```
import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;

@Entity
@Cacheable
@Cache(
    region = 'REGION_NAME',
    usage =
    CacheConcurrencyStrategy.READ_ONLY|READ_WRITE|NONSTRICT_READ_WRITE|TRANSACTIONAL
)

public class MyClass implements Serializable { ... }
```

#### 5. Start Hibernate

You are now ready to build, deploy, and run your Hibernate application, which will also launch GemFire. See the [Hibernate documentation](#) for further information about performing these actions.

#### 6. Verify that GemFire Started Successfully

Similar to Hibernate, GemFire uses Simple Logging Facade for Java (SLF4J) to log messages. Upon successful startup, GemFire will log the following message:

```
2010-11-15 INFO
[com.gemstone.gemfire.modules.hibernate.GemFireRegionFactory] -
    <Initializing GemFire Modules Version 1.1>
```

SLF4J can send this and other GemFire log messages to several logging frameworks. Refer to the [SLF4J](#) website for additional information.

## Advanced Configuration (Hibernate Cache Module)

After you set up Hibernate to work with the GemFire module, GemFire will run automatically with preconfigured settings. Depending on your topology, you may want to change these settings.

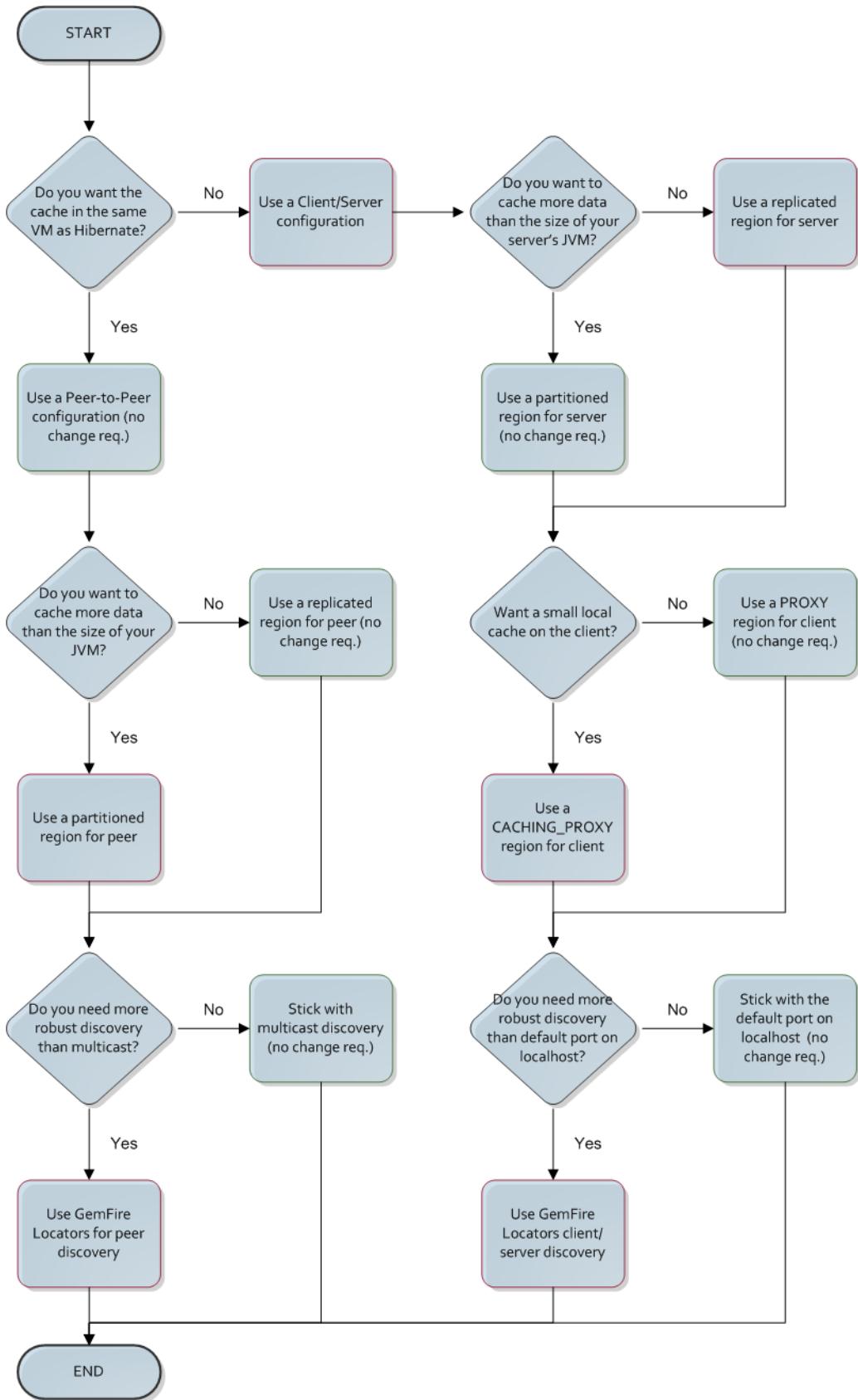
GemFire has these default settings:

- GemFire runs in a peer-to-peer configuration, rather than a client/server configuration.
- GemFire processes are discovered using a default multicast channel.
- GemFire "locators", which provide discovery and load balancing services, are not used for member discovery.
- The cache region is replicated.

These settings might not reflect your preferred usage. To change these and other settings, refer to this section.

### Configuration Flowchart

The following flowchart illustrates some of the decisions you might consider before making configuration changes. If the action says "no change req.", then no configuration change is required to run GemFire in that mode. All other actions are cross-referenced below the flowchart. For more information about the common GemFire topologies, refer to [Common GemFire Topologies](#) on page 806. For general information on how to make configuration changes, refer to "Changing GemFire's Default Configuration".

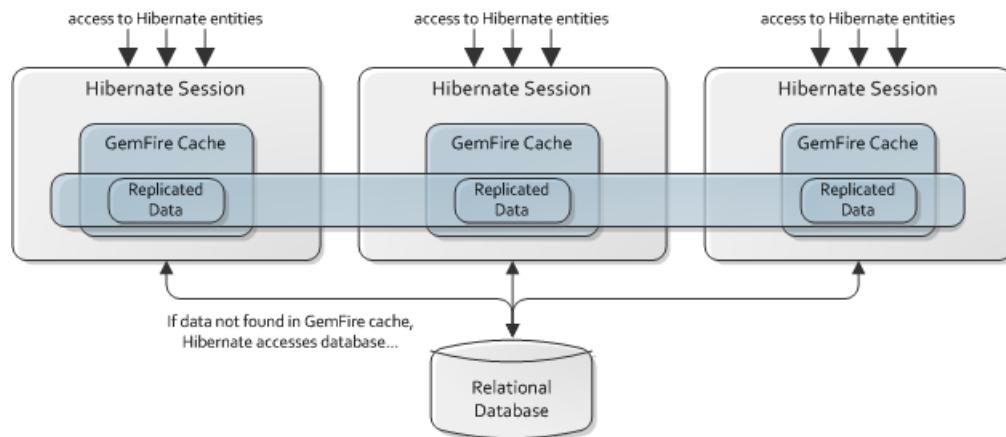


- **Use a Client/Server configuration:** See [Changing the GemFire Topology](#) on page 807 for more information.
  - **Use a replicated region for server:** Change the server region to "REPLICATE\_HEAP\_LRU" as described in [Changing Client/Server Region Attributes](#) on page 809.
  - **Use a partitioned region for peer:** Change the peer region to "PARTITION" as described in [Changing Peer-to-Peer Region Attributes](#) on page 808.
  - **Use a CACHING\_PROXY region for client:** Change the client region to "CACHING\_PROXY" as described in [Changing Client/Server Region Attributes](#) on page 809.
  - **Use GemFire Locators for peer discovery:** See [Using GemFire Locators with a Peer-to-Peer Configuration](#) on page 808 for more information.
  - **Use GemFire Locators for client/server discovery:** See [Using GemFire Locators with a Client/Server Configuration](#) on page 809 for more information.

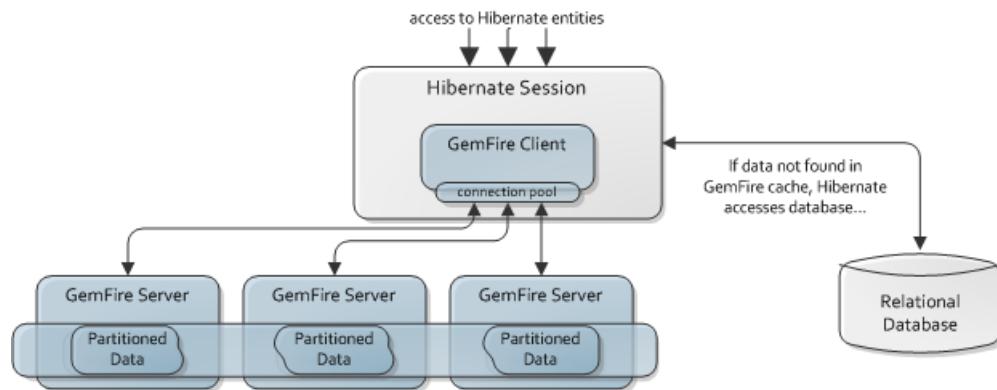
## Common GemFire Topologies

Before configuring the GemFire module, you must consider which basic topology is suited for your usage. The configuration process is slightly different for each topology. For information about GemFire license options as they relate to your desired configuration, refer to [Choosing a License Option Based on Topology](#) on page 46.

- **Peer-to-Peer Configuration.** In a peer-to-peer configuration, each GemFire instance within a Hibernate JVM contains its own cache. It communicates with other GemFire instances as peers (rather than clients and servers). Depending on the way the GemFire data region is defined, data is either replicated across all GemFire peers or it is partitioned so that each instance's cache contains a portion of the entire data set. By default, data is replicated. A peer-to-peer configuration is useful when multiple systems need fast access to all data. This configuration is also the simplest one to set up and does not require any external processes. By default, the GemFire module will operate in a peer-to-peer configuration.



- **Client/Server Configuration.** In a client/server configuration, the Hibernate JVM operates as a GemFire client, which must communicate with one or more GemFire servers to acquire data. A client/server configuration is useful when you want to separate the Hibernate JVM from the cached data. In this configuration, you can reduce the memory consumption of the Hibernate process since data is stored in separate GemFire server processes. For instruction on running GemFire in a client/server configuration, refer to [Changing the GemFire Topology](#) on page 807.



## Changing vFabric GemFire Default Configuration (Hibernate Cache Module)

To edit GemFire configuration information (such as how members locate each other), you can add GemFire properties to Hibernate's `hibernate.cfg.xml` file.

When adding properties, use the following syntax:

```
<property name="gemfire.PROPERTY_NAME">PROPERTY_VALUE</property>
```

In the preceding code snippet, `PROPERTY_NAME` is the property name and `PROPERTY_VALUE` is the property value. The list of configurable system properties include any of the properties that can be specified in GemFire's `gemfire.properties` file. The property changes associated with the most common configurations are described below.

### Changing the GemFire Topology

By default, the topology is set for a peer-to-peer configuration rather than a client/server configuration. To use the peer-to-peer configuration, you don't have to make any changes. If you want to change the configuration to a client/server topology, edit the Hibernate configuration file:

```
<property name="gemfire.cache-topology">client-server</property>
```

Since the Hibernate process operates as a GemFire client in this configuration, you must manually launch a GemFire cache server. You can do this with the following script (which should be installed into the **bin** subdirectory wherever GemFire is located):

```
In Unix:  
prompt$ ./cacheserver.sh start
```

```
In Windows:  
prompt> cacheserver.bat start
```

 **Note:** When running the cache server, make sure the Hibernate jar and the GemFire modules jar are in the classpath, along with any dependencies. You can use the `CLASSPATH` environment variable or you can set the `-classpath <classpath>` command line argument when launching the cache server.

Without making any further configuration changes, the client and server attempt to locate each other on the same system using a default port (40404). Though useful for testing purposes, you would not normally deploy a client/server configuration in this way. Refer to "Using GemFire Locators with a Client/Server Configuration" for information on creating and using Locators, which provide both discovery and load balancing services.

## Changing Peer-to-Peer Region Attributes

You can change the region attributes from within the `hibernate.cfg.xml` file using the same region shortcuts specified in [Region Shortcuts and Custom Named Region Attributes](#) on page 150.

```
<property name="gemfire.default-region-attributes-id">
    REGION_ATTRIBUTE
</property>
```

In this example, `REGION_ATTRIBUTE` can be any of the pre-defined region shortcuts. By default, this is `REPLICATE_HEAP_LRU`. Other available region shortcuts include: `REPLICATE`, `REPLICATE_PERSISTENT`, `PARTITION`, `PARTITION_PERSISTENT`, `PARTITION_REDUNDANT`, `PARTITION_REDUNDANT_PERSISTENT`. If you plan to configure a region for persistence, refer to [Turning On Persistence](#) on page 810.

You can also control the region type on a per-entity basis using the following code:

```
<property name="gemfire.region-attributes-for: com.foo.Bar">
    REGION_ATTRIBUTE
</property>
```

In this example, `com.foo.Bar` is the name of your class and `REGION_ATTRIBUTE` can be one of the region shortcuts.

## Using a Different Multicast Port with a Peer-to-Peer Configuration

For a GemFire peer-to-peer member to communicate on a different multicast port than the default (10334), use the `mcast-port` property.

```
<property name="gemfire.mcast-port">10445</property>
```

This example uses port 10445 as the multicast port.

## Using GemFire Locators with a Peer-to-Peer Configuration

By default, GemFire peers discover each other using multicast communication on a known port (as specified by the `mcast-port` property). Alternatively, you can configure member discovery using one or more dedicated locators. A locator provides both discovery and load balancing services.

To run GemFire with one or more locators instead of a multicast port, use the `locators` property.

```
<property name="gemfire.locators">host1[port1],host2[port2]</property>
```

In this example, `host1` and `host2` are the hostnames (or IP addresses) for each of the locators; `port1` and `port2` are the ports for each of the locators.

Although this code will tell the peer to look for a locator, it does not automatically start the locator. To launch a locator automatically within the same application as the one running the Hibernate process, use this code, with your hostname (or IP address) and port settings:

```
<property name="gemfire.start-locator">host[port]</property>
```

Alternatively, you can start a locator in a separate process using the `gemfire` command-line tool found in the `bin` subdirectory wherever GemFire is located.

In Unix:

```
./gemfire.sh start-locator -port=10334
```

In Windows:

```
gemfire.bat start-locator -port=10334
```

This example starts a locator that listens on port 10334.

## Changing Region Names

By default, the GemFire Hibernate module puts entities in a region whose name is the fully qualified path of the entity. The module uses these region names to eagerly prefetch related entities. Therefore, we recommend that you do not change the default names of the regions that are created by Hibernate.

## Changing Client/Server Region Attributes

When using a client/server configuration, the default region attribute for the GemFire servers is a PARTITION region. This means that the data is partitioned across the pool of GemFire servers. You can modify this behavior using the same instructions specified in [Changing Peer-to-Peer Region Attributes](#) on page 808.

In addition to the region attribute of the servers, you can also modify the region attributes for the GemFire client, which is the process running Hibernate. By default this region is set to PROXY. This means that the client will not cache data.

You can change the client region attributes from within the hibernate.cfg.xml file using the same client region shortcuts specified in [Region Shortcuts and Custom Named Region Attributes](#) on page 150.

```
<property name="gemfire.default-client-region-attributes-id">
  CLIENT_REGION_ATTRIBUTE
</property>
```

In this example, CLIENT\_REGION\_ATTRIBUTE can be any of the pre-defined region shortcuts. By default, this is PROXY. If you would like the client to keep a local cache of recently used data, change the region to CACHING\_PROXY.

You can also control the client region type on a per-entity basis using the following code:

```
<property name="gemfire.client-region-attributes-for: com.foo.Bar">
  CLIENT_REGION_ATTRIBUTE
</property>
```

In this example, com.foo.Bar is the name of your class and CLIENT\_REGION\_ATTRIBUTE can be one of the client region shortcuts.

## Using GemFire Locators with a Client/Server Configuration

By default, GemFire clients and servers discover each other on a pre-defined port (40404) on the localhost. This is not typically the way you would deploy a client/server configuration. A preferred solution is to use one or more dedicated locators. A locator provides both discovery and load balancing services.

To run GemFire with one or more locators, use the `locators` property.

```
<property name="gemfire.locators">host1[port1],host2[port2]</property>
```

In this example, host1 and host2 are the hostnames (or IP addresses) for each of the locators; port1 and port2 are the ports for each of the locators.

Although this code will tell the **client** to look for a locator, it does not automatically start the locator. You can start a locator using the `gemfire` command-line tool found in the `bin` subdirectory wherever GemFire is located.

In Unix:

```
./gemfire.sh start-locator -port=10334
```

In Windows:

```
gemfire.bat start-locator -port=10334
```

This example starts a locator that listens on port 10334.

You also need to tell the GemFire **server** to use a locator when you launch the server. You can do this through a command-line argument when you start up the server with the script provided in the `bin` subdirectory wherever GemFire is located:

In Unix:

```
prompt$ ./cacheserver.sh start locators=hostname[port]
```

In Windows:

```
prompt> cacheserver.bat start locators=hostname[port]
```

If you are running more than one locator, use a comma-separated list of locators. Refer to [vFabric GemFire Servers](#) on page 741 for more information on using this script.



**Note:** When running the cache server, make sure the Hibernate jar and the GemFire modules jar are in the classpath, along with any dependencies. You can use the `CLASSPATH` environment variable or you can set the `-classpath <classpath>` command line argument when launching the cache server.

## Turning On Persistence

If you change the region type to any one of the persistent modes (such as `PARTITION_PERSISTENT` and `REPLICATE_PERSISTENT`), cache data will get stored into the current working directory. To change this location, supply disk storage information in GemFire's `cache.xml` configuration file. For example:

```
<cache>
    <!-- define a disk store with a couple of directories. -->
    <!-- All data logs are stored here -->
    <disk-store name="CacheDataRepository">
        <disk-dirs>
            <disk-dir>c:\cache_data</disk-dir>
            <disk-dir dir-size="20480">d:\cache_data</disk-dir>
        </disk-dirs>
    </disk-store>

    <!-- now change the predefined region attributes to use this disk store
-->
    <region-attributes id="MYAPP_PARTITION_PERSISTENT"
        ref-id="PARTITION_PERSISTENT">
        <disk-store>CacheDataRepository</disk-store>
    </region-attributes>
</cache>
```

This example creates a 20480 MB data repository in the `d:\cache_data` subdirectory. It also defines a region attribute called `MYAPP_PARTITION_PERSISTENT`, which is derived from the `PARTITION_PERSISTENT` region definition.

You must also change Hibernate's `hibernate.cfg.xml` file to point to this xml file and to reference this particular region definition:

```
<property name="gemfire.default-region-attributes-id">
    MYAPP_PARTITION_PERSISTENT
</property>
<property name="gemfire.cache-xml-file">
    c:\gemfire_cache.xml
</property>
```

## Caching Query Results

In a Hibernate application, the caching of query results is recommended for queries that are run frequently with the same parameters. However, most applications using Hibernate do not benefit from query results caching because Hibernate will invalidate cached results when even a single entity involved in the query is updated. For

example, suppose that a query on the Person entity is cached. As soon as any instance of Person is updated, created or deleted, Hibernate will invalidate this query because it cannot predict the impact of the change on the cached query results. Therefore, the GemFire Hibernate Cache Module does not distribute query results to other GemFire member caches. The query results are always cached only in the GemFire cache local to the Hibernate session.

### Remapping the Hibernate EnumType with the GemFire EnumType

Due to an issue with Hibernate in a client-server topology, application classes must be present on the GemFire cache server when a Hibernate application uses annotations. If you do not want to place the application classes on the GemFire cache server, you can modify the Hibernate configuration to use the `com.gemstone.gemfire.modules.hibernate.EnumType` in place of `org.hibernate.type.EnumType`.

To configure Hibernate to use the GemFire EnumType, modify `hibernate.cfg.xml` to add a property for the mapping. For example:

```
<property name="myEnum">
  <type name="com.gemstone.gemfire.modules.hibernate.EnumType">
    <param name="enumClass">com.mycompany.MyEnum</param>
  </type>
</property>
```

## Using Hibernate Cache Module with HTTP Session Management Module

This section describes how to set up the Hibernate Cache module if you are also using the HTTP Session Management module.

If you have already configured your tc Server or Tomcat application server to use the HTTP Session Management Module, follow these steps to configure Hibernate module for your application:

1. Install and configure the Hibernate Cache Module as described in [Installing the Hibernate Cache Module](#) on page 802 and [Setting Up the GemFire Hibernate Cache Module](#) on page 802.
2. Make sure that your Hibernate application is using the same GemFire topology as your application server.
3. Include `gemfire-modules-<version>.jar` in your application .war file. This is typically done by putting the .jar file under the `WEB-INF/lib` directory.



**Note:** Even though both modules include a `gemfire-modules-<version>.jar` file in their distribution, you only need to put one `gemfire-modules-<version>.jar` file in your .war file. It does not matter which one you use. You can even be using different versions of the HTTP Session Management and Hibernate Cache modules.

4. Make sure that `gemfire.jar` is not included in your application .war file.
5. Deploy your application .war file.
6. These configuration steps will work with the default `gemfire` properties configurations that ship with the GemFire modules. If you have added or modified Gemfire properties for the HTTP Session Management Module (for example, as described in [Changing the Default GemFire Configuration in the Tomcat Module](#) on page 779 or [Changing the Default GemFire Configuration in the tc Server Module](#) on page 769), then you must make the same changes for the Hibernate Cache module. See [Changing vFabric GemFire Default Configuration \(Hibernate Cache Module\)](#) on page 807 for instructions on how to modify the default Hibernate Cache configuration.



**Note:** Using the Hibernate Cache Module with the HTTP Session Management Module creates a single GemFire cache in your application server. This cache will be shared by all applications deployed on the application server. Therefore, if two applications use the same entity objects (specified by fully qualified class names) or are configured to use the same cache region, all instances of these entities will end up in

the single GemFire cache region. If this is not the desired behavior, then include gemfire.jar in the application war file. A GemFire cache will be created in the application's class loader.

# Chapter 59

---

## Gemcached

Gemcached is a vFabric GemFire adapter that allows Memcached clients to communicate with a GemFire server cluster, as if the servers were memcached servers. Memcached is an open-source caching solution that uses a distributed, in-memory hash map to store key-value pairs of string or object data.

For information about Memcached, see <http://www.memcached.org>.

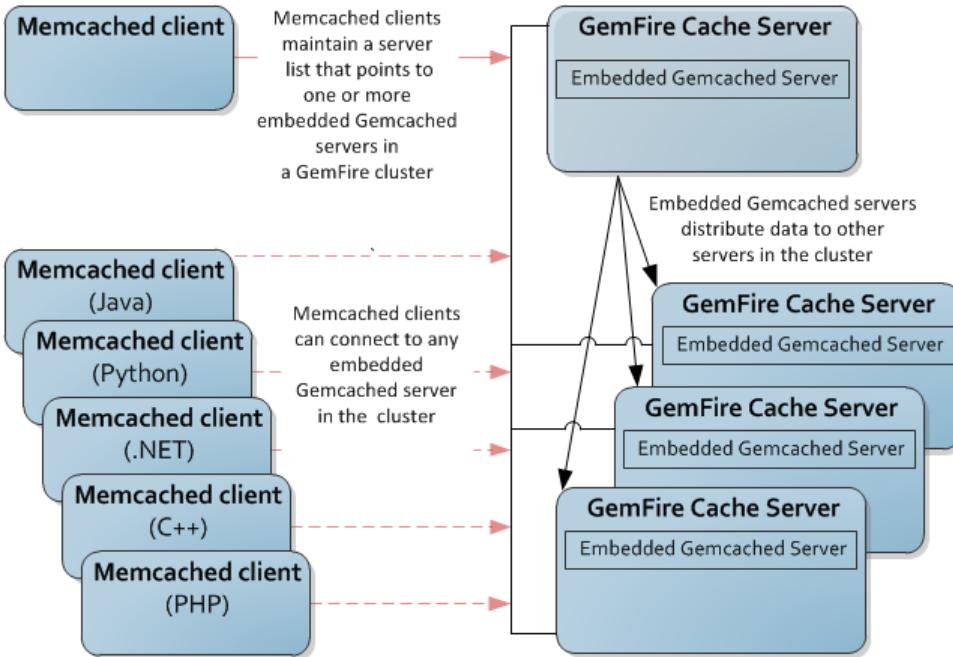
### How Gemcached Works

Applications use memcached clients to access data stored in embedded Gemcached servers.

Applications can use memcached clients that are written in Python, C#, Ruby, PHP, and other programming languages. Each memcached server in a cluster stores data as key/value pairs. A memcached client maintains a list of these servers, determines which server has the required data, and accesses the data directly on that server.

To integrate memcached with vFabric GemFire, you embed a Gemcached server within a GemFire cache server. These *Gemcached* servers take the place of memcached servers. The memcached client uses its normal wire protocol to communicate with the Gemcached servers, which appear to the client as memcached servers. No code changes in the clients are needed. GemFire manages the distribution and access to data among the embedded Gemcached servers.

As shown in [Figure 2: Gemcached Architecture](#) on page 814, memcached clients, which ordinarily maintain a list of memcached servers, now maintain a list of embedded Gemcached servers. If more embedded Gemcached servers are added to the cluster, the new servers automatically become part of the cluster. The memcached clients can continue to communicate with the servers on the list, without having to update their list of servers.



**Figure 2: Gemcached Architecture**

Memcached clients use the memcached API to read and write data that is stored in memcached servers; therefore, client-side GemFire features are not available to these clients. Gemcached servers, however, can use GemFire's server-side features and API. These features include the following. (For more detail, see [Advantages of Gemcached over Memcached](#) on page 816.)

- Data consistency and scalability.
- High availability.
- Read-through, write through, and write behind to a database, implemented from within the distributed GemFire cache.
- Storage keys and values of any type and size.
- For applications, a choice among partitioned and replicated region configurations.
- Automatic overflow of data to disk in low-memory scenarios.
- Efficient persistence of data to disk.
- Data replication across WANs or multiple networks.
- Configurable expiration of cached data.
- Configurable eviction of cached data.

## Deploying and Configuring a Gemcached Server

You can configure and deploy Gemcached servers in a Java class or by using the gfsh command-line interface.

The following sections describe how to configure and deploy an embedded Gemcached server. You can configure and start a GemCached server either by invoking a Java class that calls the cache server's `start()` method, or by starting the cache server using the gfsh command line.

### Embedding a Gemcached server in a GemFire Java Application

The `com.gemstone.gemfire.memcached` package contains a single class, `GemFireMemcachedServer`. (See the [GemFire Javadocs](#).) Use this class to configure and embed a Gemcached server in a GemFire cache

server. For example, the following statement creates and starts an embedded Gemcached server on port number 5555 using the binary protocol:

```
GemFireMemcachedServer server =
    new GemFireMemcachedServer(5555, Protocol.BINARY);server.start();
```



**Note:** By default, Gemcached servers use the ASCII protocol.

When you start a Gemcached server, by default, it creates a RegionShortcut.PARTITION region named gemcached where data used by memcached clients is stored. You can alter these defaults by configuring the region using the `cache.xml` or `gemfire.properties` files. See [Distributed System and Cache Configuration](#) on page 133.

### Starting a Gemcached Server Using a gfsh Command

You can also start a Gemcached server with the gfsh command-line interface. Use the following syntax:

```
gfsh>start server
--name=<server_name>
--server-port=<port_number>
--memcached-port=<port_number>
--memcached-protocol=BINARY|ASCII
```



**Note:**

You can also set the memcached port number and protocol in the `gemfire.properties` file.

If the `memcached-port` property is not specified, the embedded Gemcached server is not started.

### Configuring a Gemcached Server with the `gemfire.properties` File

You can set the following properties in the `gemfire.properties` file that are used when starting Gemcached servers:

**Table 51: Gemcached Properties**

Property	Description
memcached-port	The port number where the Gemcached server listens for connections from memcached clients.  If the port number is set to 0 or the <code>memcached-port</code> parameter is omitted, the Gemcached server does not start.
memcached-protocol	Memcached supports both ASCII and binary communication protocols. (See <a href="#">Memcached protocol</a> ) By default, Gemcached uses the ASCII protocol.  Set one of the following values: <ul style="list-style-type: none"> <li>• ASCII (default)</li> <li>• BINARY</li> </ul>

## Advantages of Gemcached over Memcached

The standard memcached architecture has inherent architectural challenges that make memcached applications difficult to write, maintain, and scale. Using Gemcached with vFabric GemFire addresses these challenges.

The standard memcached architecture has inherent architectural challenges that make memcached applications difficult to write, maintain, and scale. Using Gemcached with vFabric GemFire addresses these challenges.

**Data consistency.** Memcached clients must maintain a list of servers where the distributed data is stored. Each client must maintain an identical list, with each list ordered in the same way. It is the responsibility of the application logic to maintain and propagate this list. If some clients do not have the correct list, the client can retrieve stale data. In GemFire clusters, all members communicate with each other to maintain data consistency, which eliminates the need to code these behaviors in the memcached clients.

**High availability.** When a memcached server becomes unavailable, memcached clusters are subject to failures or degraded performance because clients must directly query the backend database. Memcached-based applications must be coded to handle these failures, while GemFire clusters handle such failures natively.

**Faster cluster startup time.** When a memcached cluster fails and a restart is required, the data must be reloaded and distributed to the cluster members while simultaneously processing requests for data. These startup activities can be time-consuming. When a GemFire cluster restarts, data can be reloaded from other in-memory, redundant copies of the data or from disk, without having to query the back end database.

**Better handling of network segmentation.** Large deployments of memcached can use hundreds of servers to manage data. If, due to network segmentation, some clients cannot connect to all nodes of a partition, the clients will have to fetch the data from the backend database to avoid hosting stale data. GemFire clusters handle network segmentation to ensure that client responses are consistent.

**Automatic scalability.** If you need to add capacity to a memcached cluster, you must propagate a new server list to all clients. As new clients come on line with the new list, older clients may not have a consistent view of the data in the cluster, which can result in inconsistent data in the servers. Because new GemFire cache server members automatically discover each other, memcached clients do not need to maintain a complete server list. You can add capacity simply by adding servers.

**Scalable client connections.** A memcached client may need to access multiple pieces of data stored on multiple servers, which can result in clients having a TCP connection open to every server. When a memcached client accesses a Gemcached server, only a single connection to a Gemcached server instance is required. The Gemcached server manages the distribution of data using GemFire's standard features.

# Chapter 60

---

## DiskConverterPre65to65

The vFabric GemFire DiskConverterPre65to65 utility converts disk files with persisted region and event queue data from versions earlier than GemFire 6.5 into the GemFire 6.5 format.



**Note:** DiskConverterPre65to65 does not work with version 5.8 input files.

### Running DiskConverterPre65to65

Use the DiskConverterPre65to65 to convert disks used in GemFire versions earlier than 6.5 to vFabric GemFire 6.5.



**Note:** The DiskConverterPre65to65 tool described in this topic converts files from GemFire versions earlier than 6.5 to a format that is compatible with vFabric GemFire 6.5. If you are upgrading to vFabric GemFire 7.0, you need to perform an additional disk upgrade using the `upgrade offline-disk-store` command in `gfsh` to convert vFabric GemFire 6.5 compatible disks to GemFire 7.0.

The DiskConverterPre65to65 tool is provided in the GemFire product installation `DiskConverterPre65to65/bin` directory in a Linux shell script, `DiskFileConverter.sh`.

DiskConverterPre65to65 tool takes input and output `cache.xml` files, with disk persistence specifications, and old and new GemFire version specifications.

At an operating system prompt, enter this command line:

```
./DiskFileConverter.sh [OLD_CACHE_XML=old_version_cache_XML_file]  
[CACHE_XML=new_version_cache_XML_file]  
[OLD_GEMFIRE=old_version_gemfire_product_location]  
[GEMFIRE=new_version_gemfire_product_location]  
[CLASSPATH=classpath_specification]
```

Example, run from the product's `DiskConverter` directory:

```
bin/DiskFileConverter.sh OLD_CACHE_XML=examples/e2_601.xml  
NEW_CACHE_XML=examples/e2_65.xml OLD_GEMFIRE=/export/GemFire/gemfire601  
NEW_GEMFIRE=/export/GemFire/gemfire6511
```

Provide the `cache.xml` files and GemFire versions at the command line, or set them as environment variables. For example, you could set the environment variable `OLD_CACHE_XML` and not explicitly provide the `OLD_CACHE_XML` specification at the command line.

The disk file converter only uses the elements of your `cache.xml` files that pertain to disk persistence. For example, cache writer and listener specifications are not relevant and are ignored. So you could provide partial

files to the converter tool. If you provide partial files, make sure to provide specifications for all information that was persisted to disk in the older version. This includes any persisted gateway sender event queue contents and the regions that use the gateway sender for WAN distribution. The most simple way to run the tool is with the complete old version and new version `cache.xml` files.

For more information on the `DiskConverterPre65to65`, including examples, see the `README.txt` file in the `DiskConverterPre65to65` directory.

# Chapter 61

---

## GemFire Pulse

GemFire Pulse is a Web Application that provides a graphical dashboard for monitoring vital, real-time health and performance of GemFire clusters, members, and regions.

Use Pulse to examine total memory, CPU, and disk space used by members, uptime statistics, client connections, WAN connections, and critical notifications. Pulse communicates with a GemFire JMX manager to provide a complete view of your GemFire deployment. You can drill down from a high-level cluster view to examine individual members and even regions within a member, to filter the type of information and level of detail.

By default, GemFire Pulse runs in a Tomcat server container that is embedded in a GemFire JMX manager node. You can optionally deploy Pulse to a Web application server of your choice, so that the tool runs independently of your GemFire clusters. Hosting Pulse on an application server also enables you to use SSL for accessing the application.

See [GemFire Pulse System Requirements](#) on page 7.

### Pulse Quick Start (Embedded Mode)

Use Pulse in embedded mode to monitor a GemFire deployment directly from a GemFire JMX Manager. By default, the embedded Pulse application connects to the local JMX Manager that hosts the Pulse application. Optionally, configure Pulse to connect to a GemFire system of your choice.



**Note:** GemFire cannot start Pulse in embedded mode if you run the GemFire member itself inside of a Tomcat server instance. GemFire displays an error message if you attempt to start an embedded Pulse application using this configuration. See [Hosting Pulse on a Web Application Server](#) on page 820 instead.

To run Pulse in embedded mode:

1. Configure a GemFire member to run as a JMX Manager node, specifying the HTTP port on which you will access the Pulse Web application. For example, the following command starts a GemFire locator as a JMX Manager node, using the default HTTP port 8080 for the Pulse application:

```
gfsh  
gfsh> start locator --name=loc1
```



**Note:** GemFire locators become JMX Manager nodes by default. To start a non-locator member as a JMX Manager node, include the `--J=-Dgemfire.jmx-manager=true` option. To specify a non-default port number for accessing the Pulse application, include the `--J=-Dgemfire.jmx-manager-http-port=port_number` option when starting the JMX Manager node.

When the JMX Manager node boots, it starts an embedded Tomcat instance and deploys the Pulse Web application at the specified or default HTTP port.

`gfsh` automatically connects to the manager when you start it in this way. If you already started a manager process earlier, use the `connect` command in `gfsh` to connect to that process.

- Access the embedded Pulse application from a Web browser. If you are connected to the GemFire cluster using `gfsh`, use the `start pulse` command to load the correct URL in your browser:

```
gfsh> start pulse
```

Or, enter the URL `http://address:jmx-manager-http-port/pulse` directly in your Web browser, substituting the address and HTTP port of the manager. For example, you access Pulse on the local locator machine from Step 1 at the URL `http://localhost:8080/pulse`.

- If you have configured authentication for the Pulse application, enter the username and password of a valid Pulse account in the login screen. Otherwise, enter the default "admin" in both fields. Click **Sign In** to continue.

See [Configuring Pulse Authentication](#) on page 822.

- After you log in, Pulse displays the main cluster view for the local distributed system. See [Using Pulse Views](#) on page 822.



**Note:** When running in embedded mode, the Pulse application connects only to the JMX Manager running in the locator or member that hosts Pulse. This enables you to monitor all members of that distributed system. You can also view (but not monitor) connected WAN clusters, and can view gateway senders and receivers that are configured in the local cluster.

## Hosting Pulse on a Web Application Server

Host Pulse on a dedicated Web application server to make the Pulse application available at a consistent address, or to use SSL for accessing the Pulse application. When you host Pulse in this way, you also configure Pulse to connect to a specific locator or JMX Manager node for monitoring.

To host Pulse on a Web application server:

- Set the `jmx-manager-http-port` property to zero (`-Dgemfire.jmx-manager-http-port=0`) when you start your GemFire JMX Manager nodes. Setting this property to zero disables the embedded Web server for hosting the Pulse application.
- Create a `pulse.properties` file somewhere in the classpath of your Web application server. For example, if you are hosting Pulse on Tomcat, create the `pulse.properties` file in the `$TOMCAT_SERVER/lib` directory.
- Define the following configuration properties in the `pulse.properties` file:

Property	Description
<code>pulse.useLocator</code>	Specify "true" to configure Pulse to connect to a GemFire Locator member, or "false" to connect directly to a JMX Manager.  When Pulse connects to a GemFire locator, the locator provides the address and port of an available JMX Manager to use for monitoring the distributed system. In most production deployments, you should connect Pulse to a locator instance; this allows Pulse to provide monitoring services using any available JMX Manager.  If you specify "false," Pulse connects directly to a specific JMX Manager. If this manager is not available, the Pulse connection fails, even if another JMX Manager is available in the distributed system.
<code>pulse.host</code>	Specify the DNS name or IP address of the GemFire locator or JMX Manager machine to which Pulse should connect. You specify either a locator or JMX Manager address depending on how you configured the <code>pulse.useLocator</code> property.

Property	Description
pulse.port	<p>Specify the port number of the GemFire locator or the HTTP port number of the JMX Manager to which Pulse should connect. You specify either a locator or JMX Manager port depending on how you configured the <code>pulse.useLocator</code> property.</p> <p>If you configured <code>pulse.useLocator=false</code>, then <code>pulse.port</code> must correspond to the <code>jmx-manager-http-port</code> setting of the JMX Manager.</p>
pulse.jmxUserName	<p>If you configured authentication for the GemFire JMX Manager node, specify a valid JMX user name that the Pulse application will use to authenticate to the JMX Manager.</p> <p> <b>Note:</b> The JMX account that Pulse uses must have both read and write privileges.</p> <p>See <a href="#">Configuring a JMX Manager</a> on page 435 for information about configuring authentication for JMX Manager nodes.</p>
pulse.jmxUserPassword	Specify the password of the JMX user account to use for authentication at startup.

For example, with this configuration Pulse connects to the locator at mylocator[10334] and accesses any available JMX Manager:

```
pulse.useLocator=true
pulse.host=locsrv.gemstone.com
pulse.port=10334
pulse.jmxUserName=pulseapp
pulse.jmxUserPassword=pulsepass
```

With this configuration Pulse accesses only the JMX Manager instance at manager1[8080]:

```
pulse.useLocator=false
pulse.host=jmxsrv.gemstone.com
pulse.port=8080
pulse.jmxUserName=pulseapp
pulse.jmxUserPassword=pulsepass
```

4. (Optional.) Configure authentication for the Pulse Web application using the instructions in [Configuring Pulse Authentication](#) on page 822.
5. Deploy the Pulse Web application to your application server. GemFire installs the `pulse.war` file in the `tools/Pulse` subdirectory of your GemFire installation directory. Depending on your application server, you may need to copy the `pulse.war` file to a deployment directory or use a configuration tool to deploy the file.
6. Access the Pulse application using the address, port, and application URL that you configure in your Web application server. For the default Tomcat URL is `http://address:8080/pulse`. Your application server provides options for configuring the address, port, and application name; substitute the correct items to access the deployed Pulse application.

Pulse connects to the locator or JMX Manager that you configured in the `pulse.properties` file, authenticating using the credentials that you configured in the file.

7. If you have configured authentication for the Pulse application, enter the username and password of a valid Pulse account in the login screen. Otherwise, enter the default "admin" in both fields. Click **Sign In** to continue.

See [Configuring Pulse Authentication](#) on page 822.

8. After you log in, Pulse displays the main cluster view for the distributed system to which it has connected. See [Using Pulse Views](#) on page 822.

## Configuring Pulse Authentication

Pulse requires all users to authenticate themselves before they can use the Pulse Web application. If you have configured JMX authentication on the GemFire JMX Manager node, the Pulse Web application itself may also need to authenticate itself to the GemFire JMX Manager node on startup.

### Authenticating Pulse Users

By default, Pulse uses the Spring security framework to implement a simple, file-based authentication mechanism. You define the names and passwords for valid Pulse users in a `pulse-users.properties` file, which must be available in the classpath of the Pulse application. By default, GemFire installs a default `pulse-users.properties` in the `WEB-INF/classes` subdirectory of the `pulse.war` file. Each line in this file defines the username, password, and access level for a Pulse user with the format:

```
username=password,role,{enabled | disabled}
```

The `role` entry must correspond to a valid Spring security role. For example, this entry shows the default "admin" user enabled with basic user access:

```
admin=admin,ROLE_USER,enabled
```

To define your own Pulse users, create a `pulse-users.properties` file with one account definition per line, and place the file in the classpath of your Pulse application before you deploy it. Only users that are specified in the `pulse-users.properties` can log into Pulse.

### Authenticating Pulse to the JMX Manager

If you run Pulse in embedded mode, the Pulse application runs on the JMX Manager node and no JMX authentication is required. You do not need to specify valid JMX credentials to start an embedded Pulse application.

If you host Pulse on a Web Application server (non-embedded mode) and you configure JMX authentication on the GemFire manager node, then the Pulse Web application must authenticate itself with the manager node when it starts. Specify the credentials of a valid JMX user account in the `pulse.properties` file, as described in [Hosting Pulse on a Web Application Server](#) on page 820.



**Note:** The credentials that you specify must have both read and write privileges in the JMX Manager node. See [Configuring a JMX Manager](#) on page 435.

## Using Pulse Views

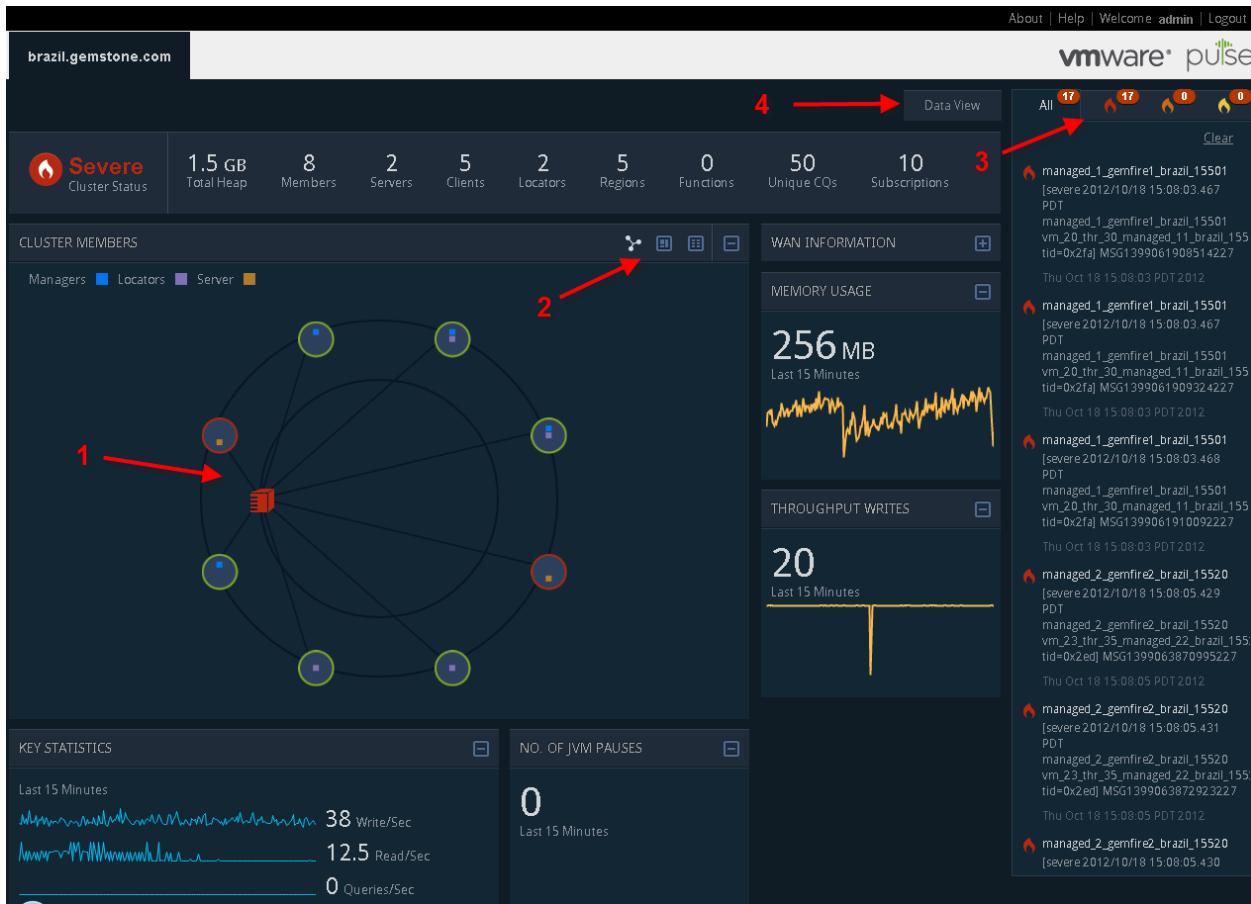
Pulse provides a variety of different views to help you monitor GemFire clusters, members, and regions.

The following sections provide an overview of the main Pulse views:

- [Cluster View](#) on page 822
- [Member View](#) on page 823
- [Data View](#) on page 824

### Cluster View

The cluster view is a high-level overview of the GemFire distributed system. It is displayed immediately after you log into Pulse. Information displays around the perimeter of the cluster view show statistics such as memory usage, JVM pauses, and throughput.

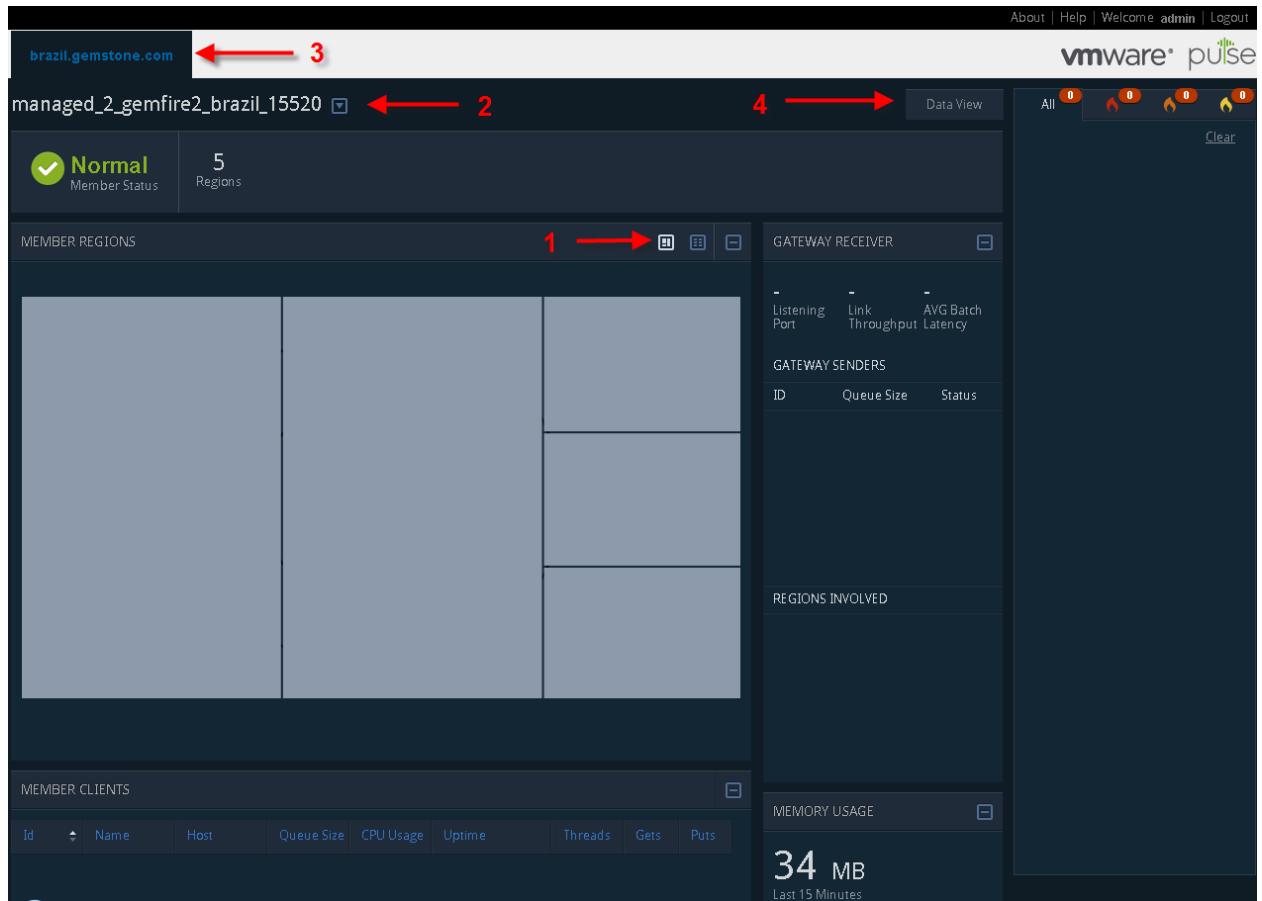


Use these basic controls while in Cluster view:

1. Click a host machine icon to display the GemFire members on that machine.
2. Click the display icons to display the GemFire members using icon view, block view, or table view.  
While in block view or table view, click the name of a GemFire member to monitor that member. See [Member View](#) on page 823.
3. Click the severity tabs to filter the message display by the level of severity.
4. Click **Data View** to display region information for the entire cluster. See [Data View](#) on page 824.

## Member View

When you select an individual GemFire member, Pulse displays the regions available on the member, as well as member-specific information such as the configured listen ports and WAN configuration.



Use these basic controls while in Member View:

1. Click the display icons to display regions using block view or table view.
2. Search for specific members by name.
3. Click the main cluster tab to return to Cluster View. See [Cluster View](#) on page 822.
4. Click **Data View** to display region information for the entire cluster. See [Data View](#) on page 824.

### **Data View**

The Pulse Data View provides a comprehensive overview of all regions in the GemFire distributed system:

**REGIONS SUMMARY - BY REGION SIZE**

**TestPartition\_4**

**PARTITION**

Type	4	6.5	10	0	12765	OFF
Members						
Write/Sec.						
Read/Sec.						
Empty Nodes						
Entry Count	12765					
Persistence						

**MEMORY USAGE**

12765 MB Used	36666 MB Total
34.81%	

**REGION MEMBERS**

Search

- managed\_1\_gemfire1\_brazil\_
- managed\_2\_gemfire2\_brazil\_
- managed\_2\_gemfire1\_brazil\_
- managed\_1\_gemfire2\_brazil\_

**READS** Last 15 Minutes

**WRITES** Last 15 Minutes

Use these basic controls while in Data View:

1. Click a region to display more information about that region.
2. Click the display icons to display regions using block view or table view.
3. Search for specific members that host the selected regions.
4. Click the main cluster tab to return to Cluster View. See [Cluster View](#) on page 822.



# Chapter 62

---

## DataBrowser

You connect to a vFabric GemFire distributed system with the vFabric GemFire DataBrowser utility, and use it to query the data interactively with OQL ad-hoc queries.

Use DataBrowser to perform the following tasks:

- Execute ad-hoc queries on any data region on any cache node in the distributed system. The DataBrowser displays the results in a tabular format that allows you to drill to down into complex nested objects.
- Configure the maximum number of rows returned by any query to avoid excessive resource use on the server.
- Simultaneously execute multiple continuous queries on one or more cache servers in the distributed system. The DataBrowser continuously updates the results in a tabular window.
- Provide security credentials when connecting to a distributed system through a plug-in mechanism.

### DataBrowser System Requirements

Verify that your system meets the installation and runtime requirements for DataBrowser.

DataBrowser is designed to run on Microsoft Windows and Red Hat Linux. DataBrowser requires the Java Runtime Environment (JRE) 1.6.0 or higher to run.

You must use the vFabric GemFire DataBrowser executable included with GemFire 7.0 to connect to vFabric GemFire 7.0 distributed systems. You cannot use past versions of vFabric GemFire DataBrowser to connect to a GemFire 7.0 system.

The minimum recommended hardware is a machine with Intel® 2 GHz Pentium® 4 (or equivalent) and 2 GB RAM.

In the DataBrowser startup scripts, you may want to set the JVM heap size explicitly to a value suitable for the typical result sizes returned by the queries that you run in your distributed system.

### Installation and Licensing

Starting in vFabric GemFire 7.0, DataBrowser is distributed with vFabric GemFire. The DataBrowser tool is installed when you install GemFire.

### Installation and Licensing Requirements

Make sure your system meets the requirements described in [DataBrowser System Requirements](#) on page 827.

When the DataBrowser connects to a GemFire locator or JMX Manager, it does not require any licensing. However, when the tool connects to a vFabric GemFire cache server to perform queries, the tool connects to the server a client and will consume a client license if you are not using an unlimited-client-upgrade license. See [Understanding vFabric GemFire Licenses](#) on page 45 for more information.

## Installing DataBrowser

Locate DataBrowser in your GemFire installation:

```
<product_directory>/tools/DataBrowser
```

Where *<product\_directory>* corresponds to the location where you installed GemFire. If you installed using RPM, the default location is `/opt/vmware/vfabric_gemfire/vFabric_GemFire_XX` where XX corresponds to the version of GemFire (for example, `vFabric_GemFire_70`) that you have installed

The DataBrowser tool installation has three subdirectories:

- **bin.** DataBrowser launch scripts, `databrowser.bat` and `databrowser`.
- **docs.** `index.html` page that links to the latest GemFire documentation (requires an Internet connection).
- **lib.** Jars and binary libraries needed to run DataBrowser .

## Uninstalling DataBrowser

To uninstall DataBrowser, delete the DataBrowser folder and its contents from your GemFire installation, or delete the entire GemFire product tree if you are uninstalling GemFire.

## Running DataBrowser

Run the installed DataBrowser with a running vFabric GemFire system.

DataBrowser connects directly to a running GemFire JMX Manager or GemFire Locator to run its queries.

The DataBrowser connects to a running JMX Manager to discover the data regions and cache servers in a GemFire distributed system and display them in the user interface. Queries must be executed by selecting data region, and optionally, a member that is hosting the region. If no member is indicated, the DataBrowser automatically selects the first region host that it finds.



**Note:** The DataBrowser does not guard against executing complex queries such as those that do a join across regions on a cache server that does not have all the regions required by the query.

To run the DataBrowser tool, perform the following steps:

1. Make sure that at least a JMX Manager or a Locator has been started and is running in the GemFire distributed system that you want to connect to and query. See [Starting a JMX Manager](#) on page 433 or [vFabric GemFire Locator Process](#) on page 737 for information on starting or viewing status of JMX Manager or Locators.
2. Add any application classes in the CLASSPATH so that DataBrowser can deserialize objects that it receives from queries. These application classes can also be set in the DataBrowser's preferences.
3. Since the DataBrowser creates a temporary directory upon startup, make sure you have write permissions to the DataBrowser directory.
4. You can start the DataBrowser either by executing the DataBrowser scripts directly or by starting the tool through the gfsh console.

To execute DataBrowser using the scripts, run one of the following scripts (depending on your OS) located in the `<product_directory>/Tools/DataBrowser/bin` directory:

- **Windows**

```
prompt>databrowser.bat
```

- **Unix/Linux**

```
databrowser
```

When starting DataBrowser directly through the DataBrowser script, you may want to explicitly set the JVM heap size to a value suitable for the result sizes that you expect for your queries. Set the desired JVM heap size using your JAVA\_ARGS variable.

To start the DataBrowser using the gfsh console:

1. Start gfsh. See [Starting gfsh](#) on page 641 if you're new to gfsh.
2. Connect to a running Locator or JMX Manager node.



**Note:** If you do not have a running Locator or JMX Manager node, start one. See [start locator](#) Starts a locator. The command creates a subdirectory and log file named after the locator. .

3. At the gfsh prompt, enter the following:

```
gfsh>start data-browser
```

The DataBrowser window appears and will be automatically connected to the running Locator.

5. **For SSL connections:** If you need to connect DataBrowser to a SSL-enabled JMX Manager, locator or server, then you need to specify SSL authentication when starting up DataBrowser.

Use SSL authentication for DataBrowser by setting the JAVA\_ARGS environment variable. For example:

#### Unix/Linux:

```
JAVA_ARGS=${JAVA_ARGS} \
-Djavax.net.ssl.keyStoreType=jks \
-Djavax.net.ssl.keyStore=/path/to/trusted.keystore \
-Djavax.net.ssl.keyStorePassword=password \
-Djavax.net.ssl.trustStore=/path/to/trusted.keystore \
-Djavax.net.ssl.trustStorePassword=password;
export JAVA_ARGS
```

#### Windows:

```
set JAVA_ARGS="%JAVA_ARGS% ^
-Djavax.net.ssl.keyStoreType=jks ^
-Djavax.net.ssl.keyStore=/path/to/trusted.keystore ^
-Djavax.net.ssl.keyStorePassword=password ^
-Djavax.net.ssl.trustStore=/path/to/trusted.keystore ^
-Djavax.net.ssl.trustStorePassword=password
```

## Connecting DataBrowser to a vFabric GemFire System

In DataBrowser, you must connect to a Locator or directly to a JMX Manager in a vFabric GemFire distributed system to perform DataBrowser queries.

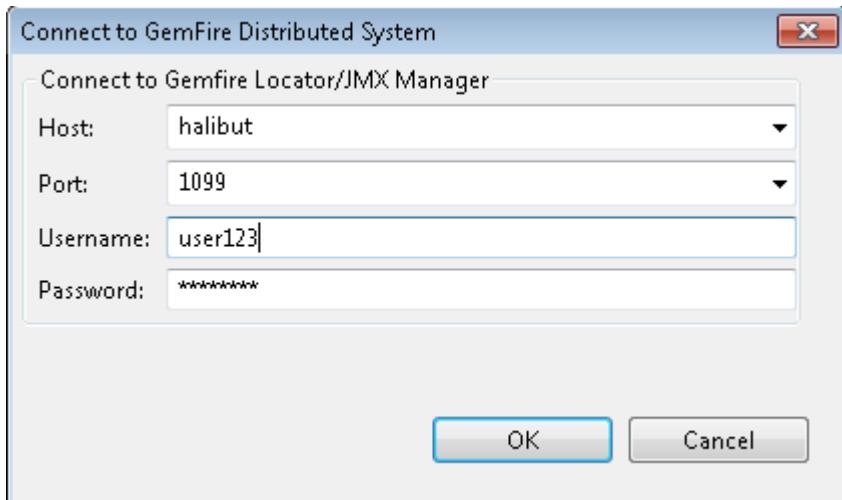
After you start DataBrowser, you connect to a GemFire distributed system to run queries. To connect to GemFire, perform the following steps.



**Note:** If you started DataBrowser through the gfsh prompt, gfsh automatically connects you to a JMX Manager node if one is up and running. To connect to a different JMX Manager node or Locator, select the Disconnect on the toolbar.

To connect DataBrowser (if you are not already connected to a GemFire Distributed System), click on the toolbar item labeled **Connect to GemFire...** or select **File, Connect to GemFire...** from the menu bar.

The **Connect to GemFire Distributed System** login screen appears:



The host and port fields refer to that of the RMI enabled JMX Manager or Locator in the GemFire distributed system to which you want to connect.

These host port combinations are saved by the DataBrowser to allow you to easily select a previous JMX Manager or Locator from the drop down lists in this dialog. You can set the number of entries in this list in the Connection Preferences dialog.



**Note:** RMI does not allow reserved characters in the host name, including '%', '\$', and '-'. For a complete list of reserved characters consult RFC 952. If your host names contain non-compliant characters, we recommend using the IP address instead.

The Username and Password fields correspond to a username and password that is listed in the password file on the JMX Manager. The name of this password file is specified in the `jmx-manager-password-file` GemFire property. See [Setting Up JMX Authentication for GemFire Management and Monitoring](#) on page 504 for more information. Once connected to the JMX Manager, the DataBrowser can fetch information on which GemFire regions are present in the distributed system, as well as the members on which they are located.

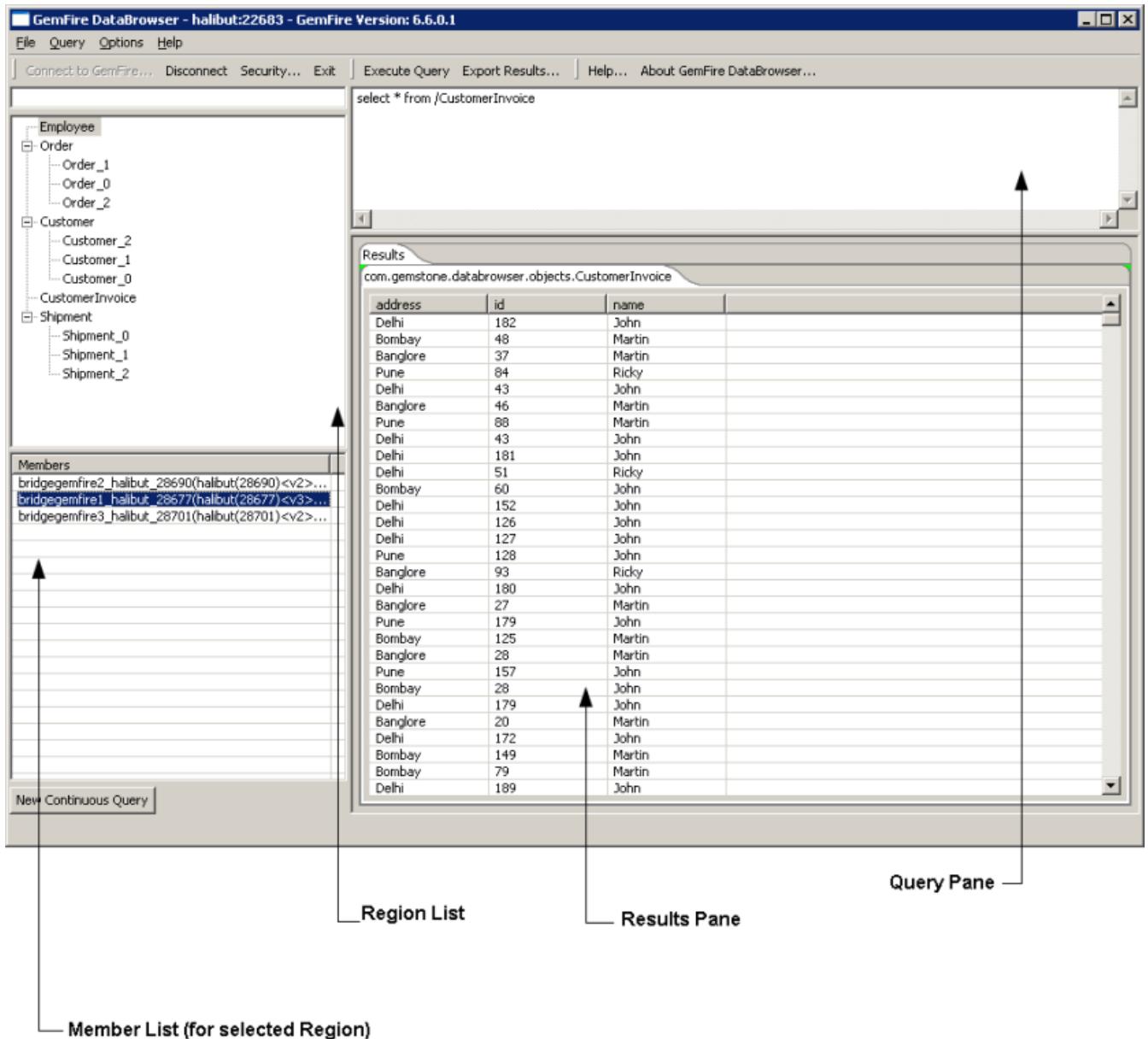
To perform queries, the DataBrowser must then be able to connect as a client to a GemFire cache server. You configure these security settings in the [DataBrowser Preferences](#) on page 833 of the tool.

## Tour of the DataBrowser Interface

To work in DataBrowser, you need to understand the screen layout and the options the browser provides.

### Main Window

DataBrowser displays the Main Window at startup. All DataBrowser functionality is available through this window.



## Toolbar

Access most functionality offered by the DataBrowser, including the ability to connect to and disconnect from a distributed system; specify security properties; execute queries; and export results to an XML file.

## Menu

Set your preferences for the application. The menu displays the Preferences dialog that allows you to set your preferences for connections made to distributed systems, query execution, security preferences, and other miscellaneous options such as additional classpath components for the application, location of the log file, logging level among others.

## Region List

The region list shows the regions that are present in the distributed system. This information is refreshed according to the interval specified in the application preferences. The list is displayed as a tree where the child nodes represent child regions of a GemFire regions.

## Member List

When a region is selected the GemFire members that have the region defined are displayed in the member list. Right-clicking on a member displays a context menu where you can execute a Continuous Query (CQ) on the member for the selected region.

## Query Pane

The query pane is a multi-line editor where you can enter OQL queries for execution. If a line is selected, the text on that line is the query string used for execution. If no line is selected, the text of the last line is automatically used as the query string for execution.

## Results Pane

The results pane displays each results item in the table in the pane. Multiple tabs may be displayed under the results pane, each tab representing a specific Java type returned in the results.

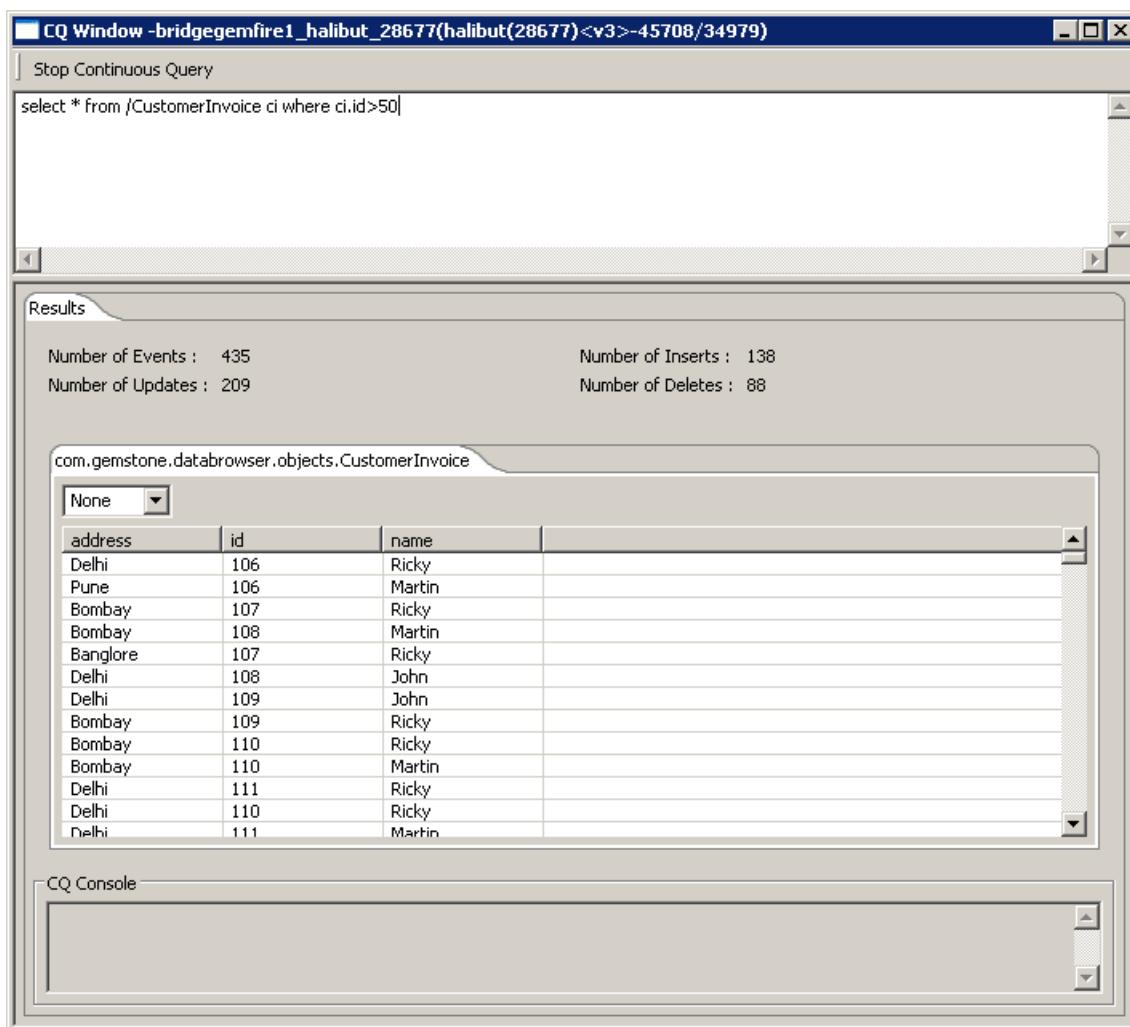
## Continuous Query (CQ)

A Continuous Query is syntactically identical to a regular GemFire OQL query, except that once its is registered on the member, the member continues sending result items that match the query specifications as changes happen in the region on the member.

The Continuous Query window has a single item in its toolbar that allows you to stop the execution of the CQ. It also shows some additional information about the query as it is being executed on the member. This includes the total number of events that have occurred on the member since the CQ was registered, including inserts, updates and deleted.

Each Results tab in a ContinuousQuery window has a drop down list that contains the field (columns in the table) names of the items in the results. This is useful in filtering the results. Selecting a field in this list allows you to filter the results displayed in the table by having it display only unique values for that field. For example, if one of the fields in the query results is a zip code, applying a filter on that field will result in the table displaying only a single row for each unique zip code returned by the query.

There is also an embedded pane that shows any messages or exceptions returned by the member.



## Exporting Results to XML

You can export the contents of the Results pane in the Main Window to an XML file. The export also creates an associated XSD. This allows you to capture the results and manipulate it and examine it further using other tools at a later point in time.

## Explore Functionality

The Results Pane in both the Main window and the CQ window has a context menu applicable to a single table row that allows you to drill down into the record in the result set. This is particularly useful to examine a record that contains fields that are collections or composite types.

To view custom application types, you must include the jars in the CLASSPATH or specify their location in the DataBrowser preferences.

## DataBrowser Preferences

Customize how DataBrowser works for you through the Preferences dialog.

User specified preferences can be set in the Preferences dialog, accessible through on the menu bar by selecting **Options** and then **Preferences...**.

## Connection

These are the Connection options:

- Number of entries to keep in history list of JMX Managers/Locators to which you have connected.
- The timeout, in milliseconds, to be used on connection attempts to the JMX Manager or Locator.
- The server response timeout, which is the amount of time, in milliseconds, for DataBrowser to wait for a response from a server. This property is the same as the client pool attribute read-timeout

## Query

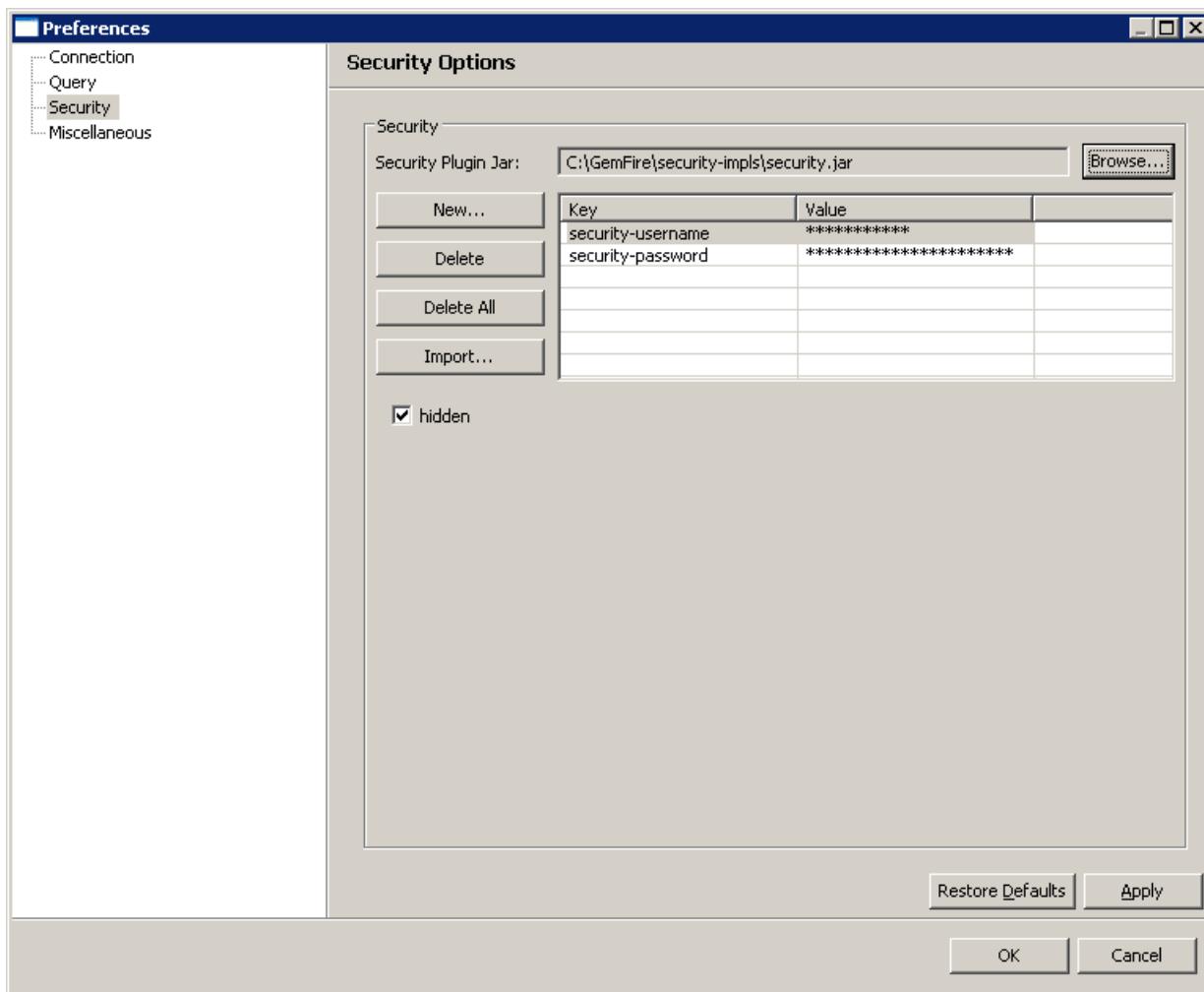
Use this option to automatically append a specified LIMIT clause to each query, if one is not already present. This can help avoid out of memory situations in cases where the heap size of the VM used by the DataBrowser is not sufficient when the query results in extremely large data volumes being returned. This preference setting does not apply to CQs.

## Security

Use the Security preferences to provide a jar that implements the GemFire security plug-in interface, and to define properties to the interface for executing queries on a cache server.

The 'hidden' checkbox toggles the masked state of the values provided. Optionally, you can import properties from a property file by clicking on the 'Import' button.

The 'Security' group box, with security-specific UI controls, is also available in a pop-up when you click on the 'Specify Security..' button on the Main Window toolbar.



## Miscellaneous

With the Miscellaneous settings you can specify the following:

- Logging options. These are applied when you restart DataBrowser. These are the logging options:
  - Location of the log files.
  - The maximum size of a log file. When a log file reaches this threshold, the log file is rolled over to a new log file.
  - Number of rolled-over log files to store. When rolling over to a new log takes the total number of log files over this limit, the oldest log file is deleted.
- The rate at which the DataBrowser updates its view of regions and members in the distributed system.
- Custom application jars to add to the DataBrowser's classpath.

## How DataBrowser Works

DataBrowser interacts with vFabric GemFire both as a GemFire client and as an RMI client of the JMX Manager.

### Discovery

As an RMI client, DataBrowser uses JMX to get information about the data regions in the distributed system and the servers on which they are located. This information is displayed on the left side of the Main Window,

where the data regions are displayed in a hierarchical tree, with child data regions displayed as sub-nodes of their parent data region. The lower pane on the left shows the list of cache servers on which the region selected in the upper list is located.

### Query Execution

A region must be selected to execute a query specified in the Query Pane. You may also select a cache server on which the query will be executed. If no cache server is selected, then the first one listed in the Member Pane is chosen for the execution of the query. DataBrowser does not prevent execution of complex queries, such as those creating a join across regions, on a cache server that does not contain the regions. In this case an error message, sent by query engine on the server is displayed.

DataBrowser creates a connection pool to a cache server the first time a query is executed on it. This pool uses the default settings specified by GemFire. This connection pool is maintained until the DataBrowser application is shut down, or the DataBrowser received a notification that the member is no longer present in the distributed system either because it has left or crashed. Though only a single ad-hoc query can be executed at a time, multiple CQs may be executed simultaneously. If these queries are being executed on the same cache server, then multiple connections to the cache server are automatically created.

DataBrowser uses the default settings of connection pools. These settings allow it to start with a single connection and allow for unbounded number of connections in pool. The DataBrowser maintains a connection pool per cache server, with a single connection in the pool. There is no fail-over to another cache server in case a connection is lost. If a query is being currently executed on a cache server that leaves, an error is returned in the UI.

### Security

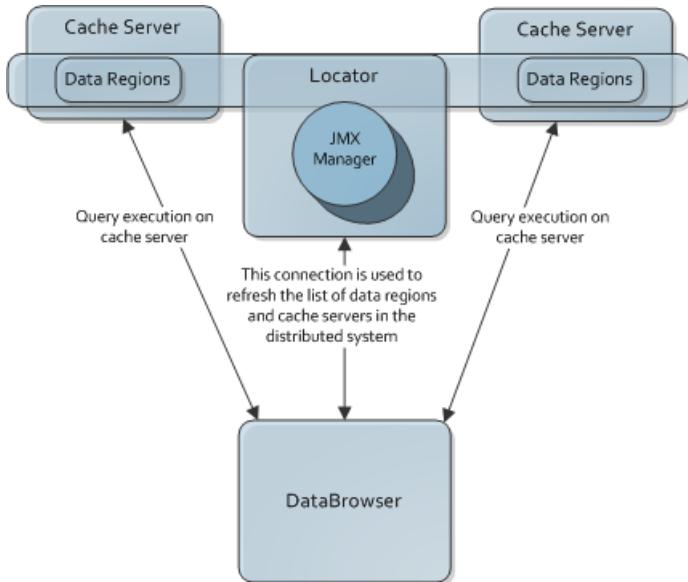
DataBrowser supports the GemFire security mechanism. It attempts to execute a query without presenting any security credentials to the cache server. If the distributed system is security enabled, then the DataBrowser received a security exception. It handles this by presenting a popup dialog populated with the name of the security plugin jar and properties specified in the Preferences dialog. The user may modify these if needed, or simply click the OK button to have the DataBrowser use the default jar and the properties. Currently DataBrowser does not provide mechanism that filters the list of regions and cache servers displayed based on the security credentials.



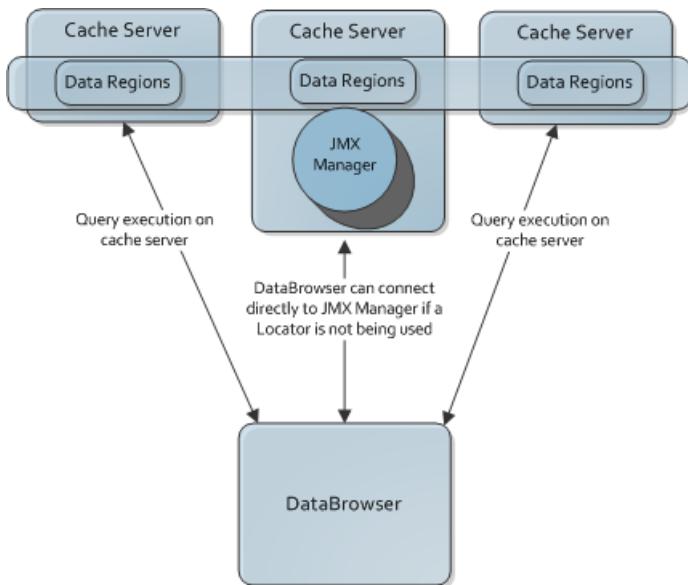
**Note:** For details on GemFire security, see the [Security](#) on page 497.

### Connection by DataBrowser to GemFire Distributed System

You can connect to a GemFire Distributed System through a Locator:



Or you can connect directly to a JMX Manager node if you are not using a Locator:



For details on client connection pools, see the [Client/Server Configuration](#) on page 177 in the *vFabric GemFire User's Guide*.



# Chapter 63

---

## Visual Statistics Display

The Visual Statistics Display (VSD) utility reads GemFire statistics and produces graphical displays for analysis. VSD is included in your GemFire product distribution in the `tools/vsd` subfolder.

### VSD Overview

VSD helps you monitor the performance of vFabric GemFire and diagnose performance problems.

For specific information on the statistics produced by vFabric GemFire, see the product documentation.

Your GemFire product creates a statistical archive file named `filename.gfs`. The file logs useful statistics — counters and gauges that describe the state of the system at a particular moment in time. The file collects statistics at specific sampling intervals, which you can set at various levels to monitor different types of behavior.

The VSD tool reads the sampled statistics and produces graphical displays for analysis. Typically, the points on a line of a VSD graph represent the values for a particular statistic over time. VSD's extensive online help offers more complete reference information about the tool.

The following screenshots of the VSD tool display statistics and a graph analysis of selected statistics.

Main    File    Chart    Template    Help

File: 1. gemfire66\_main/cq/cqserver Linux 2.6.35-28-generic i386 ketchum.gemstone.com GemFire 6.6.0.1 #shabit 32842 as of 08/25/2011 09:38:41 PDT

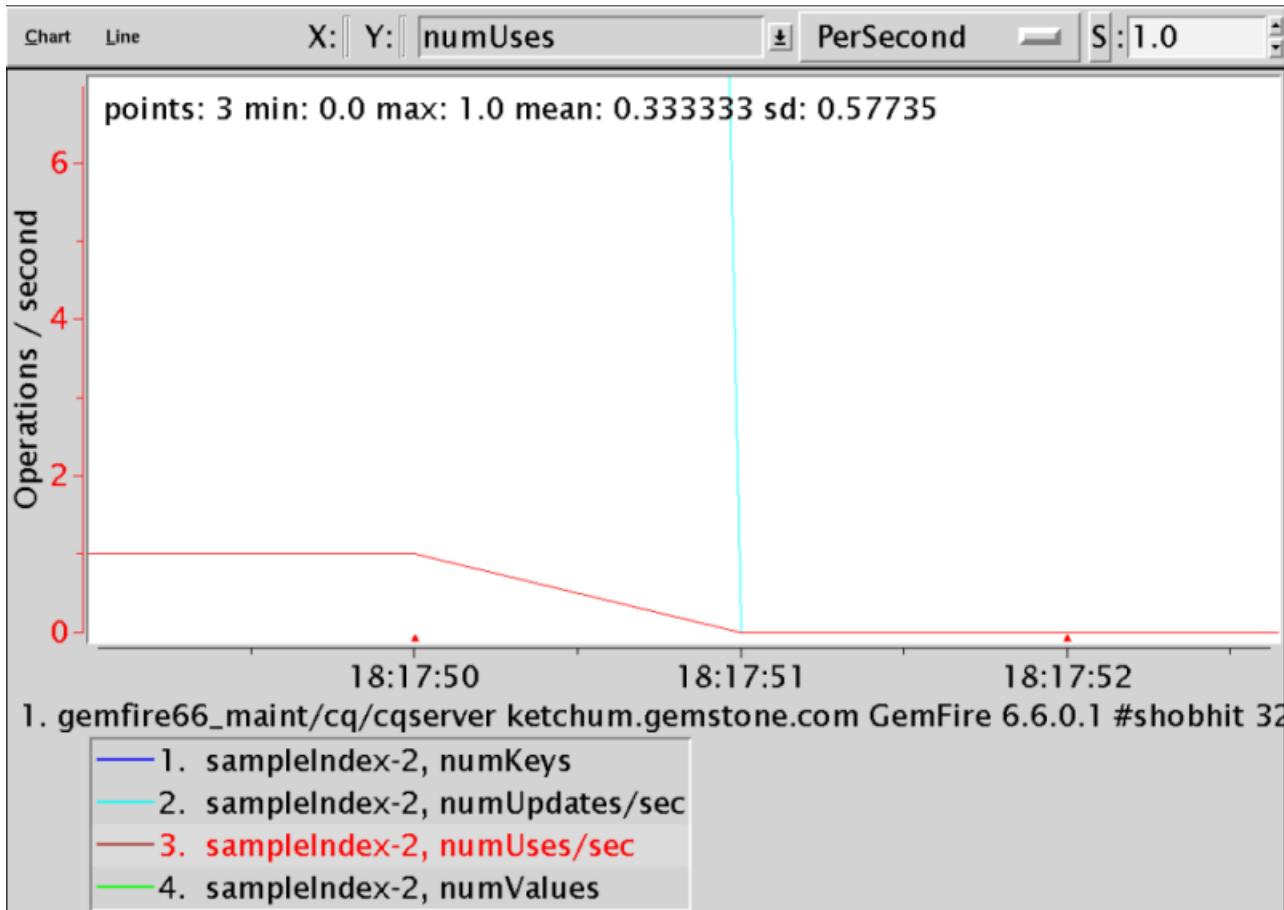
StartTime	File	Samples	Pid	Type	Name
09/02 18:17:48	1	1005	3412	VMStats	vmStats
09/02 18:17:48	1	1005	3412	VMMemoryUsageStats	vmNonHeapMemoryStats
09/02 18:17:48	1	1005	3412	VMMemoryUsageStats	vmHeapMemoryStats
09/02 18:17:48	1	1005	3412	VMMemoryPoolStats	PSSurvivorSpace-Heapmemory
09/02 18:17:48	1	1005	3412	VMMemoryPoolStats	PSPermGen-Non-heapmemory
09/02 18:17:48	1	1005	3412	VMMemoryPoolStats	PSOldGen-Heapmemory
09/02 18:17:48	1	1005	3412	VMMemoryPoolStats	PSEdenSpace-Heapmemory
09/02 18:17:48	1	1005	3412	VMMemoryPoolStats	CodeCache-Non-heapmemory
09/02 18:17:48	1	1005	3412	VMGCSStats	PSScavenge
09/02 18:17:48	1	1005	3412	VMGCSStats	PSMarkSweep
09/02 18:17:48	1	1005	7002	StatSampler	statSampler
09/02 18:17:49	1	1003	3412	ResourceManagerStats	ResourceManagerStats
09/02 18:17:49	1	1004	3412	PartitionedRegionStats	/test
09/02 18:17:49	1	1005	1980419617	LinuxSystemStats	ketchum.gemstone.com
09/02 18:17:49	1	1005	3412	LinuxProcessStats	CqServer proc
09/02 18:17:49	1	1004	3412	IndexStats	sampleIndex2
09/02 18:17:49	1	1004	3412	DLockStats	dlockStats
09/02 18:17:49	1	1005	3412	DistributionStats	distributionStats
09/02 18:17:50	1	1002	3412	CacheServerStats	ketchum.gemstone.com-0.0.0/0.0.0:5522
09/02 18:17:49	1	1004	3412	CachePerfStats	RegionStats-partitionMetaData
09/02 18:17:49	1	1004	3412	CachePerfStats	RegionStats-partition-test
09/02 18:17:48	1	1005	3412	CachePerfStats	cachePerfStats
09/02 18:17:50	1	1002	3412	CacheClientNotifierStatistics	cacheClientNotifierStats

```

numKeys
numUpdates
numUses
numValues
updatesInProgress
updateTime-IndexStats
usesInProgress
useTime

```

Chart:



## VSD System Requirements

View a list of platforms that are known to work with VSD.

The following platforms are known to work with the current VSD release:

- RedHat 3
- RedHat 4
- RedHat 5
- RedHat 6
- SLES 8
- SLES 9
- SLES 10
- Solaris 2.8
- Mac OSX
- Windows NT 4.0
- Windows 98/ME/2K
- Windows XP SP3
- Windows Server 2003
- Windows Server 2008
- Windows 7 (Note: Requires slight modification; see below.)

### Windows 7 Support

To use VSD on Windows 7, perform the following steps:

1. Start Windows Explorer and navigate to the `GemFireProductDir\tools\vsd\bin\` directory (where `GemFireProductDir` corresponds to the location where you installed GemFire.)
2. Right click and select properties for `vsd.bat`.
3. Select the Compatibility tab.
4. Click "Run this program in compatibility mode for" and then select Windows XP SP3.
5. Repeat steps 2 and 3 for all the other executables in the `bin` directory.

## Installing and Running VSD

Start the VSD tool, load statistics files, and maintain the view you want on your statistics.

### Install VSD

VSD is a free analysis tool and is provided as-is.

Starting in vFabric GemFire 7.0, VSD is distributed with vFabric GemFire. To install VSD, install vFabric GemFire. See [Installing vFabric GemFire](#) on page 27 for instructions.

After you install GemFire, you can find VSD in the following location of your GemFire installation:

`<product_directory>/tools/vsd`

Where `<product_directory>` corresponds to the location where you installed GemFire. If you installed using RPM, the default location is `/opt/vmware/vfabric_gemfire/vFabric_GemFire_XX` where XX corresponds to the version of GemFire (for example, `vFabric_GemFire_70`) that you have installed

The VSD tool installation has three subdirectories:

- **bin.** The scripts and binaries that can be used to run VSD on a variety of operating systems. The following scripts are included with this release:
  - `vsd`
  - `vsd.bat`

The following binaries are included with this release:

- `vsdwishSunOS` - for Solaris
- `vsdwishLinux` - for Linux
- `vsdwishWindows_NT.exe` - for Windows
- `vsdwishDarwin` - for Mac

- **lib.** The jars and binary libraries needed to run VSD

### Start VSD

To start VSD, you can either execute the scripts directly or start it through the `gfsh` interface. To start VSD using the provided scripts, enter the following command at the prompt:

- **Windows:**

`prompt> vsd`

- **Linux/Unix, MacOS or Other OS:**

`$ vsd`

•

To start VSD using `gfsh`, start a `gfsh` prompt and enter the following command:

`gfsh>start vsd`

## Load a Statistics File into VSD

You have several options for loading a statistics file into VSD:

- Include the name of one or more statistics files on the VSD command line. Example:  

```
vsd <filename.gfs> ...
```
- Browse for an existing statistics file through Main > Load Data File.
- Type the full path in the File entry box, then press Enter.
- Switch to a statistics file that you've already loaded by clicking the down-arrow next to the File entry.

After you load the data file, the VSD main window displays a list of entities for which statistics are available. VSD uses color to distinguish between entities that are still running (shown in green) and those that have stopped (shown in black).

## Maintain a Current View of the Data File

If you select the menu item File > Auto Update, VSD automatically updates your display, and any associated charts, whenever the data file changes. Alternatively, you can choose File > Update periodically to update the display manually.

## About Statistics

Some statistics are cumulative from when the GemFire system was started. Other statistics are instantaneous values that may change in any way between sample collection.

Cumulative statistics are best charted per second or per sample, so that the VSD chart is readable. Absolute values are best charted as No Filter.

## Viewing Statistics in VSD

Select statistics and view them using chart templates and customized charts.

### Statistic Levels

Each statistic has a characteristic called a level that reflects the amount of background knowledge that you would need to use the statistic with understanding. You can set up VSD to list (in its main window and in associated charts) only those statistics that are at, or below, a certain level of complexity — common, advanced, or wizard.

To establish the levels of statistics that you want to display in VSD, choose the menu item Main > Statistic Level in the main VSD window.

### Select Statistics for Viewing

1. In the VSD list, click the left mouse button to select the entity or entities you want to view.
  - Search for a specific session name or process ID. To find a specific entity, click the mouse in the process list, then press Ctrl-S. When the dialog box appears, enter the PID or name of the entity that you're looking for. VSD highlights the first entity with that PID or name. To find the next match, press Ctrl-S again. To select the highlighted item, click on it. When you're done, press Return.
  - Select all entities or by statistic or by type. Statistics are available for various entities, which are listed under the heading Type in the process list.



**Note:** You can use the right mouse button to perform these functions:

- Combine multiple entities into a single line. This can be quite helpful. For example, if you want to measure page reads per second for several hundred entities, you could select all the entities, then combine them into a single line in the chart, thus rendering the data much more readable.

- Combine multiple entities from different files into a single line.
  - Eliminate flatlines — entities whose values are always zero.
  - Select Single File mode so only one loaded file can be enabled at a time.
  - Select for created lines to have absolute timestamps, which is useful when merging files.
2. Select a statistic for viewing from the statistics list just below the process list.
  3. With the selecting statistic, do one of the following:
    - To display the statistic in a new chart, type the name of the chart in the Chart entry box, then click New Chart. (Note: If you don't explicitly specify a chart name, VSD will assign one for you.)
    - To display the statistic in an existing chart, select the chart name in the Chart entry box. Then click on Add Line.
  4. To add another statistic to the chart, repeat steps 1 through 3.

### Using VSD Chart Templates

VSD templates let you quickly add a set of lines to a chart. Templates are helpful if you find yourself performing the same task frequently in VSD — for example, monitoring the same five or six statistics. By creating a template for the statistics that you want to monitor most frequently, you can automate the task of building charts.

In your template, you can assign a filter for each statistic, to determine how much information is displayed for that statistic. You can also restrict the template to look for extreme conditions (for example, processes that are consuming 90% or more of the CPU).

VSD is shipped with a set of predefined templates, which are maintained in the `.vsdtemplates` file in your home directory.

Task	Procedure
Create a new chart from a template	In the VSD main window, choose the menu item Template > New Template Chart. This is a good way to display some of the more useful system statistics.
Apply a template to the chart you are viewing	In the Chart window, choose the menu item Chart > Add From Template. (Note: If you have zoomed in on a chart, the template filter is only applied to values within the zoomed range.)
Reread the <code>.vsdtemplates</code> template file into VSD after you've edited it	In the VSD main window, choose the menu item Template > Reload Template File.
Save the current chart as a template	In the Chart window, configure the chart as you desire, then choose the menu item Chart > Save Template. The template is saved to the <code>.vsdtemplates</code> file. If you save the current chart as a template, you may still need to edit the <code>.vsdtemplates</code> file so that you can give it a more useful name and make the information and patterns it captures more general.

### Chart Menu (Chart Window)

To customize the way VSD displays statistics in your chart, you can choose items from the Chart window's Chart menu.

Chart Menu Item	Effect
Add from Template	Expand template and add resulting lines to chart.
Save Template	Save all lines on chart as a template.
Paste	Paste last item on clipboard.

Chart Menu Item	Effect
Print	Print chart.
Snapshot	Write this chart as a graphic to <code>snapshot.gif</code> .
Help . . .	Open Help window.
Zoom In	Zoom in to improve your view of the chart. After you choose this menu item, click to select one corner of the area that you want to zoom. Move the mouse pointer to the opposite corner of the zoom area, then click again. If you have a middle mouse button, you can quickly zoom in on an area by clicking the middle mouse button over it.
Zoom Out	Zoom out by using the menu button or by right-clicking in the chart window.
Compare Two Points	Log information by comparing two points.
Compute Scale All, Unscale All	Adjust the scale of the chart. This helps you view multiple statistics on the same axis.
Show Legend	Display the legend for this chart.
Time Format	Change the format of the time displayed along the X axis.
Show Time Axis Title, Show Left Axis Title, Show Right Axis Title	Display the title alongside the respective axes.
Show Current Values	Display the current X and Y values for the selected line at the top of the chart
Show Min and Max	Display the minimum and maximum values for the selected line at the top of the chart.
Show Line Stats	Display these statistics for the selected line: the number of data samples, the min, max, mean, and standard deviation. The statistics are calculated from all of the data points on the selected line in the region defined by the graph's current X axis. (To change the region, select Zoom In or Zoom Out from the Chart menu.)
Show CrossHairs	Draw cross hairs on graph of item.
Show Grid Lines	Draw grid line on graph of item.
Close	Close chart window.

For additional information about the Chart window, choose Help from the Chart menu.

### Line Menu (Chart Window)

Customize your VSD chart display using items from the Line menu (in the Chart window). Line menu commands operate on the currently selected line. To select a line, click on it or on its entry in the chart legend. VSD highlights the selected line.

Line Menu Item	Effect
Log Info	Display a log file showing the line statistics for all data samples in the region defined by the graph's current X axis.
Log Delta	Measure the difference between two values on the selected line. Select the line before choosing this menu item then click on the two points whose difference you want to compute. VSD responds by displaying a log file showing the difference in time and value between the two points; the number of data samples in the selected line segment; and the min, max, mean, and standard deviation of those samples.
Compute Scale	Compute a scale value for the selected line that will make it visible on the current chart. You can also use the Scale entry box to manually change the scale. The default scale value is 1.

Line Menu Item	Effect
Unscale	Reverse the effect of Compute Scale.
Graph on Left Axis	Display the Y axis for the selected line to the left of the chart. Otherwise, the Y axis is displayed to the right. You can use this to view multiple lines on the same chart, by graphing large values on one axis and small values on the opposite axis.
Symbol	Select a new symbol.
Style	Select a line style for connecting points: linear (default), step, natural, quadratic.
Update	Update files used by current line.
Add Lines	Add a line to the current line.
Diff Lines	Remove a line from the current line.
Divide Lines	Divide current line by another line.
Normalize	Normalize current line.
Trim Left, Right	Trim line to left or right of a data point.
Untrim Left, Right	Undo any trim line operations.
Copy	Copy current line to clipboard.
Cut	Cut current line to clipboard.
Delete	Remove the selected line from this chart.

## Customizing Your VSD Chart

You can customize and manipulate a VSD chart in many ways:

- To select a line in the chart, click on it or on its entry in the chart legend.
- To delete a line from the chart, click the middle mouse button (if available) on its entry in the chart legend. Alternatively, select the line's entry in the chart legend and choose Line > Delete.
- To find out about a specific point in a chart, hold the mouse pointer over it.

## View Statistic Information

To view a description of the most recently selected statistic, along with information about its type, level, and default filter, go to the VSD main window, then choose the menu item Main > Show Statistic Info.

In the Statistic Information window, you can redefine the level and default filter for any VSD statistic.

- The statistic's level — common, advanced, or wizard — allows you to determine whether the statistic is displayed in the VSD statistic list.
- Whenever you add a line to a chart, the filter determines how information is displayed for the selected statistic.

Default Filter	Effect
Default	No Filter if the statistic represents a snapshot of a value. PerSecond if the statistic represents a value that always increases.
No Filter	Display the raw values for the statistic with no filtering.
PerSample	Display the difference between two consecutive samples of the statistic.
PerSecond	Display the difference between two consecutive samples of the statistic, divided by the number of elapsed seconds between the two samples.

Default Filter	Effect
Aggregate	Display a running total of per-sample deltas for the statistic. Reset to zero when the delta is zero or changes direction.

Once you have added the line to a chart, you can override its default filter by specifying a new filter from the drop-down menu at the top of the Chart window.

If you leave the Statistic Information window up as you work, it changes to reflect the current statistic. In this way, you can get a quick explanation of any statistic that you're currently examining.



# Part 10

---

## Reference

*Reference* documents vFabric GemFire properties, region attributes, the cache.xml file, and cache memory requirements.

### Topics:

- [gemfire.properties and gfsecurity.properties \(GemFire Properties\)](#)
- [cache.xml](#)
- [Region Shortcuts](#)
- [Exceptions and System Failures](#)
- [Memory Requirements for Cached Data](#)



# Chapter 64

---

## gemfire.properties and gfsecurity.properties (GemFire Properties)

You use the `gemfire.properties` settings to join a distributed system and configure system member behavior. Distributed system members include applications, the cache server, the locator, and other GemFire processes.

You can place any security-related (properties that begin with `security-*`) configuration properties in `gemfire.properties` into a separate `gfsecurity.properties` file. Placing these configuration settings in a separate file allows you to restrict access to security configuration data. This way, you can still allow read or write access for your `gemfire.properties` file.

You can specify non-ASCII text in your properties files by using Unicode escape sequences. See [Using Non-ASCII Strings in vFabric GemFire Property Files](#) on page 860 for more details.



**Note:** Unless otherwise indicated, these settings only affect activities within this distributed system - not activities between clients and servers or between a gateway sender and gateway receiver in a multi-site installation.

Setting	Definition	Default
ack-severe-alert-threshold	Number of seconds the distributed system will wait after the <code>ack-wait-threshold</code> for a message to be acknowledged before it issues an alert at severe level. A value of zero disables this feature.	0
ack-wait-threshold	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.  The waiter continues to wait. The alerts are logged in the system member's log as warnings.  Valid values are in the range 0...2147483647	15
archive-disk-space-limit	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 use is unlimited.	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.	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. The switch to asynchronous messaging lasts until this process catches up, departs, or some specified limit is reached, such as <code>async-queue-timeout</code> or <code>async-max-queue-size</code> .  To enable asynchronous messaging, the value must be set above zero. Valid values are in the range 0...60000.	0

Setting	Definition	Default
	 <b>Note:</b> This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	
async-max-queue-size	<p>Affects non-conflated asynchronous queues for members that publish to this member. This is the maximum size the queue can reach (in megabytes) before the publisher asks this member to leave the distributed system.</p> <p>Valid values are in the range 0..1024.</p>  <b>Note:</b> This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	8
async-queue-timeout	<p>Affects asynchronous queues for members that publish to this member. This is the maximum milliseconds the publisher should wait with no distribution to this member before it asks this member to leave the distributed system. Used for handling slow receivers.</p>  <b>Note:</b> This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	60000
bind-address	<p>Relevant only for multi-homed hosts - machines with multiple network interface cards. Specifies the adapter card the cache binds to for peer-to-peer communication. Also specifies the default location for GemFire servers to listen on, which is used unless overridden by the <code>server-bind-address</code>. An empty string causes the member to listen on the default card for the machine. 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.</p> <p>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.</p>	<i>not set</i>
cache-xml-file	Declarative initialization file for the member's cache.	cache.xml
conflate-events	Used only by clients in a client/server installation. This is a client-side property that is passed to the server. Affects subscription queue conflation in this client's servers. Specifies whether to conflate (true setting), not conflate (false), or to use the server's conflation setting (server).	server
conserve-sockets	<p>Specifies whether sockets are shared by the system member's threads. If 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. You can override this setting for individual threads inside your application. Where possible, it is better to set <code>conserve-sockets</code> to true and enable the use of specific extra sockets in the application code if needed.</p>  <b>Note:</b> WAN deployments increase the messaging demands on a GemFire system. To avoid hangs related to WAN messaging, always set <code>conserve-sockets=false</code> for GemFire members that participate in a WAN deployment.	true
delta-propagation	Specifies whether to distribute the deltas for entry updates, instead of the full values, between clients and servers and between peers.	true
deploy-working-dir	Working directory used when deploying JAR application files to distributed system members. This directory can be local and unique to the member or a shared resource. See <a href="#">Deploying Application JARs to vFabric GemFire Members</a> on page 747 for more information.	<small>optional</small>
disable-tcp	Boolean indicating whether to disable the use of TCP/IP sockets for inter-cache point-to-point messaging. If disabled, the cache uses datagram (UDP) sockets.	false

Setting	Definition	Default
distributed-system-id	Identifier used to distinguish messages from different distributed systems. Set this to different values for different systems in a multi-site (WAN) configuration. This is required for Portable Data eXchange (PDX) data serialization. This setting must be the same for every member in the same distributed system and unique to the distributed system within the WAN installation. -1 means no setting. Valid values are integers in the range -1...255.	-1
durable-client-id	Used only for clients in a client/server installation. If set, this indicates that the client is durable and identifies the client. The ID is used by servers to reestablish any messaging that was interrupted by client downtime.	<i>not set</i>
durable-client-timeout	Used only for clients in a client/server installation. Number of seconds this client can remain disconnected from its server and have the server continue to accumulate durable events for it.	300
enable-network-partition-detection	Boolean instructing the system to detect and handle splits in the distributed system, typically caused by a partitioning of the network (split brain) where the distributed system is running.	false
enable-time-statistics	Boolean instructing the system to track time-based statistics for the distributed system and caching. Disabled by default for performance reasons and not recommended for production environments. You must also configure <code>statistics-sampling-enabled</code> to true and specify a <code>statistics-archive-file</code> .	false
enforce-unique-host	Whether partitioned regions will put redundant copies of the same data in different members running on the same physical machine. By default, GemFire tries to put redundant copies on different machines, but it will put them on the same machine if no other machines are available. Setting this property to true prevents this and requires different machines for redundant copies.	false
groups	Defines the list of groups that this member belongs to. Use commas to separate group names. Note that anything defined by the roles gemfire property will also be considered a group. See <a href="#">Using Member Groups</a> on page 745 for more information.	<i>not set</i>
jmx-manager	If true then this member is willing to be a jmx-manager. All the other jmx-manager properties will be used when it does become a manager. If this property is false then all other jmx-manager properties are ignored.	false (except on locators)
jmx-manager-access-file	By default the jmx-manager will allow full access to all mbeans by any client. If this property is set to the name of a file then it can restrict clients to only being able to read mbeans; they will not be able to modify mbeans. The access level can be configured differently in this file for each user name defined in the password file. For more information about the format of this file see Oracle's documentation of the com.sun.management.jmxremote.access.file system property. Ignored if jmx-manager is false or if jmx-manager-port is zero.	<i>not set</i>
jmx-manager-bind-address	By default the jmx-manager when configured with a port will listen on all the local host's addresses. You can use this property to configure what ip address or host name the jmx-manager will listen on. In addition, if the embedded http server is started, it will also bind to this address if it is set. Ignored if jmx-manager is false or jmx-manager-port is zero.	<i>not set</i>
jmx-manager-hostname-for-clients	Lets you control what hostname will be given to clients that ask the locator for the location of a jmx manager. By default the ip address that the jmx-manager reports is used. But for clients on a different network this property allows you to configure a different hostname that will be given to clients. Ignored if jmx-manager is false or jmx-manager-port is zero.	<i>not set</i>
jmx-manager-httpp-port	If non-zero then when the jmx manager is started, an embedded web server will also be started and will listen on this port. The web server is used to host the GemFire Pulse application. If you are hosting the Pulse web app in your own web server, then disable this embedded server by setting this property to zero. Ignored if jmx-manager is false.	8080
jmx-manager-password-file	By default the jmx-manager will allow clients without credentials to connect. If this property is set to the name of a file then only clients that connect with credentials that match an entry in this file will be allowed. Most JVMs require that the file is	<i>not set</i>

Setting	Definition	Default
	only readable by the owner. For more information about the format of this file see Oracle's documentation of the com.sun.management.jmxremote.password.file system property. Ignored if jmx-manager is false or if jmx-manager-port is zero.	
jmx-manager-port	The port this jmx manager will listen to for client connections. If this property is set to zero then GemFire will not allow remote client connections but you can alternatively use the standard system properties supported by the JVM for configuring access from remote JMX clients. Ignored if jmx-manager is false.	1099
jmx-manager-start	If true then this member will start a jmx manager when it creates a cache. Management tools like gfsh can be configured to connect to the jmx-manager. In most cases you should not set this because a jmx manager will automatically be started when needed on a member that sets "jmx-manager" to true. Ignored if jmx-manager is false.	false
jmx-manager-update-rate	The rate, in milliseconds, at which this member will push updates to any JMX Managers. Currently this value should be greater than or equal to the statistic-sample-rate. Setting this value too high will cause stale values to be seen by gfsh and GemFire Pulse.	2000
jmx-manager-ssl	If true and jmx-manager-port is not zero then the JMX Manager will only accept SSL connections. Note that the <code>ssl-enabled</code> property does not apply to the JMX Manager, but the other SSL properties do. This allows SSL to be configured for just the JMX Manager without needing to configure it for the other GemFire connections. Ignored if jmx-manager is false.	false
license-application-cache	Specifies the serial number this distributed system member will use unless a <code>license-data-management</code> serial number is also specified. This distributed member will attempt to activate a GemFire Application Cache Node license which allows it to be a peer-to-peer application cache node. If a <code>license-data-management</code> serial number is also provided, then the <code>license-application-cache</code> serial number may be specified as the license for cache clients connecting to this node. The keyword "dynamic" may be used to specify that the member will attempt to acquire a license from either a vFabric serial number file or a vFabric License Server.	<i>not set</i>
license-data-management	Specifies the serial number(s) this distributed system member will use. This distributed member will attempt to activate a GemFire Data Management Node license which allows it to provide advanced data management services. This includes hosting a cache server to which cache clients may connect. If you have serial numbers for the Unlimited Client Upgrade or Global WAN Upgrade, then list them after the serial number for Data Management Node separating each with a comma. If you do not have the Unlimited Client Upgrade, then you should also use <code>license-application-cache</code> to specify the serial number for the cache clients that will connect to this cache server. The keyword "dynamic" may be used to specify that the member will attempt to acquire a license from either a vFabric serial number file or a vFabric License Server.	<i>not set</i>
license-server-timeout	Specifies the maximum number of milliseconds to wait for a license from a vFabric License Server when attempting to dynamically acquire a license. This timeout is observed when using the keyword "dynamic" for either of the licensing properties <code>license-data-management</code> or <code>license-application-cache</code> .  Valid values are in the range of 1000...3600000	10000
license-working-dir	The writable directory where this member persists runtime information about licensing. Keep this directory the same between runs for each member so the member application can read what it persisted on the last run.	member's current working directory as determined by <code>System.getProperty("userdir")</code>
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.	<i>not set</i>

Setting	Definition	Default
	<p>For each locator, provide a host name and/or address (separated by ‘@’, if you use both), followed by a port number in brackets. Examples:</p> <pre>locators=address1[port1],address2[port2]</pre> <pre>locators=hostName1@address1[port1],hostName2@address2[port2]</pre> <pre>locators=hostName1[port1],hostName2[port2]</pre> <p> <b>Note:</b> 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.</p> <p>If you have values specified for the <code>locators</code> property, the <code>mcast-port</code> property defaults to 0.</p>	
log-disk-space-limit	Maximum size in megabytes of all inactive log files combined. 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 use is unlimited.	0
log-file	<p>File to which a running system member writes log messages. If set to null, the default is used.</p> <p>Each member type has its own default output:</p> <ul style="list-style-type: none"> <li>• application: standard out</li> <li>• locator: <code>locator.log</code></li> <li>• cacheserver: <code>cacheserver.log</code></li> </ul>	null
log-file-size-limit	Maximum size in megabytes of a log file before it is closed and logging rolls on to a new (child) log file. If set to 0, log rolling is disabled.	0
log-level	<p>Level of detail of the messages written to the system member’s log. Setting log-level to one of the ordered levels causes all messages of that level and greater severity to be printed.</p> <p>Valid values from lowest to highest are fine, config, info, warning, error, severe, and none.</p>	config
max-num-reconnect-tries	Used when a cache region’s membership attributes specify to disconnect from the distributed system and reconnect in an attempt to regain lost membership roles. Maximum number of times to try to reconnect to the distributed system when membership roles are missing.	3
max-wait-time-reconnect	Used when a cache region’s membership attributes specify to disconnect from the distributed system and reconnect in an attempt to regain lost membership roles. Maximum number of milliseconds to wait for the distributed system to reconnect on each reconnect attempt.	10000
mcast-address	<p>Address used to discover other members of the distributed system. Only used if mcast-port is non-zero. This attribute must be consistent across the distributed system.</p> <p> <b>Note:</b> Select different multicast addresses and different ports for different distributed systems. Do not just use different addresses. Some operating systems may not keep communication separate between systems that use unique addresses but the same port number.</p> <p>This default multicast address was assigned by IANA (<a href="http://www.iana.org/assignments/multicast-addresses">http://www.iana.org/assignments/multicast-addresses</a>). Consult the IANA chart when selecting another multicast address to use with GemFire.</p> <p> <b>Note:</b> This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication. If multicast is enabled,</p>	239.192.81.1 for IPv4 (the default IP version) FF38::1234 for IPv6

Setting	Definition	Default
	distributed regions use it for most communication. Partitioned regions only use multicast for a few purposes, and mainly use either TCP or UDP unicast.	
mcast-flow-control	Tuning property for flow-of-control protocol for unicast and multicast no-ack UDP messaging. Compound property made up of three settings separated by commas: byteAllowance, rechargeThreshold, and rechargeBlockMs.  Valid values range from these minimums: 10000,0.1,500 to these maximums: no_maximum ,0.5,60000.   <b>Note:</b> This setting controls only peer-to-peer communication, generally between distributed regions.	1048576,0.25, 5000
mcast-port	Port used, along with the mcast-address, for multicast communication with other members of the distributed system. If zero, multicast is disabled for member discovery and distribution.   <b>Note:</b> Select different multicast addresses and ports for different distributed systems. Do not just use different addresses. Some operating systems may not keep communication separate between systems that use unique addresses but the same port number.  Valid values are in the range 0..65535.   <b>Note:</b> This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.  If you have values specified for the <code>locators</code> property, the <code>mcast-port</code> property defaults to 0.	10334
mcast-recv-buffer-size	Size of the socket buffer used for incoming multicast transmissions. You should set this high if there will be high volumes of messages.  Valid values are in the range 2048.. OS_maximum.   <b>Note:</b> The default setting 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.   <b>Note:</b> This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	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.   <b>Note:</b> This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	65535
mcast-ttl	How far multicast messaging goes in your network. Lower settings may improve system performance. A setting of 0 constrains multicast messaging to the machine.   <b>Note:</b> This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	32
member-timeout	GemFire uses the <code>member-timeout</code> server configuration, specified in milliseconds, to detect the abnormal termination of members. The configuration setting is used in two ways: 1) First it is used during the UDP heartbeat detection process. When a member detects that a heartbeat datagram is missing from the member that it is monitoring after the time interval of 2 * the value of <code>member-timeout</code> , the detecting member attempts to form a TCP/IP	5000

Setting	Definition	Default
	<p>stream-socket connection with the monitored member as described in the next case.</p> <p>2) The property is then used again during the TCP/IP stream-socket connection. If the suspected process does not respond to the <i>are you alive</i> datagram within the time period specified in <code>member-timeout</code>, the membership coordinator sends out a new membership view that notes the member's failure.</p> <p>Valid values are in the range 1000..600000.</p>	
membership-port-range	<p>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.</p> <p>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. The specified range must include at least three available port numbers since three ports are required (UDP, FD_SOCK, and TCP).</p> <p>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.</p>	1024-65535
memcached-port	If specified and is non-zero, sets the port number for an embedded Gemcached server and starts the Gemcached server.	0
memcached-protocol	Sets the protocol used by an embedded Gemcached server. Valid values are <code>BINARY</code> and <code>ASCII</code> . If you omit this property, the <code>ASCII</code> protocol is used.	ASCII
name	Symbolic name used to identify this system member.	<i>not set</i>
redundancy-zone	Defines this member's redundancy zone. Used to separate member's into different groups for satisfying partitioned region redundancy. If this property is set, GemFire will not put redundant copies of data in members with the same redundancy zone setting.	<i>not set</i>
remote-locators	<p>Used to configure the locators that a cluster will use in order to connect to a remote site in a multi-site (WAN) configuration. To use locators in a WAN configuration, you must specify a unique distributed system ID (<code>distributed-system-id</code>) for the local cluster and remote locator(s) for the remote clusters to which you will connect.</p> <p>For each remote locator, provide a host name and/or address (separated by '@', if you use both), followed by a port number in brackets. Examples:</p> <pre>remote-locators=address1[port1],address2[port2]</pre> <pre>remote-locators=hostName1@address1[port1],hostName2@address2[port2]</pre> <pre>remote-locators=hostName1[port1],hostName2[port2]</pre>	<i>not set</i>
remove-unresponsive-client	When this property is set to true, the primary server drops unresponsive clients from all secondaries and itself. Clients are deemed unresponsive when their messaging queues become full on the server. While a client's queue is full, puts that would add to the queue block on the server.	false
roles	Comma-delimited list of strings specifying the membership roles that this member performs in the distributed system.	<i>not set</i>
security-*	<p>Used for authentication. Any custom properties needed by your <code>AuthInitialize</code> or <code>Authenticator</code> callbacks.</p> <p> <b>Note:</b> Any security-related (properties that begin with <code>security-*</code>) configuration properties that are normally configured in <code>gemfire.properties</code> can be moved to a separate <code>gfsecurity.properties</code> file. Placing these configuration</p>	<i>not set</i>

Setting	Definition	Default
	settings in a separate file allows you to restrict access to security configuration data. This way, you can still allow read or write access for your <code>gemfire.properties</code> file.	
security-client-accessor	Used for authorization. Static creation method returning an <code>AccessControl</code> object, which determines authorization of client-server cache operations. This 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.	<i>not set</i>
security-client-accessor-pp	Used for authorization. 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.	<i>not set</i>
security-client-auth-init	Used for authentication. Static creation method returning an <code>AuthInitialize</code> object, which obtains credentials for peers in a distributed system. The obtained credentials should be acceptable to the <code>Authenticator</code> specified through the security-peer-authenticator property on the peers.	<i>not set</i>
security-client-authenticator	Used for authentication. Static creation method returning an <code>Authenticator</code> object, which is used by a peer to verify the credentials of the connecting peer.	<i>not set</i>
security-client-dhalgo	Used for authentication. For secure transmission of sensitive credentials like passwords, you can encrypt the credentials using the Diffie-Hellman key exchange algorithm. 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.	<i>not set</i>
security-log-file	Used with authentication. The log file for security log messages. If not specified, the member's regular log file is used.	<i>not set</i>
security-log-level	Used with authentication. Logging level detail for security log messages. Valid values from lowest to highest are fine, config, info, warning, error, severe, and none.	config
security-peer-auth-init	Used with authentication. Static creation method returning an <code>AuthInitialize</code> object, which obtains credentials for peers in a distributed system. The obtained credentials should be acceptable to the <code>Authenticator</code> specified through the security-peer-authenticator property on the peers.	<i>not set</i>
security-peer-authenticator	Used with authentication. Static creation method returning an <code>Authenticator</code> object, which is used by a peer to verify the credentials of the connecting peer.	<i>not set</i>
security-peer-verifymember-timeout	Used with authentication. Timeout in milliseconds used by a peer to verify membership of an unknown authenticated peer requesting a secure connection.	1000
server-bind-address	Relevant only for multi-homed hosts - machines with multiple network interface cards. 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. This setting has no effect on locator configuration.  Specify the IP address, not the hostname, because each network card may not have a unique hostname.  An empty string causes the servers to listen on the same card used for peer-to-peer communication. This is either the <code>bind-address</code> or, if that is not set, the machine's default card.	<i>not set</i>
socket-buffer-size	Receive buffer sizes in bytes of the TCP/IP connections used for data transmission. To minimize the buffer size allocation needed 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

Setting	Definition	Default
socket-lease-time	<p>Time, in milliseconds, a thread can have exclusive access to a socket it is not actively using. A value of zero causes socket leases to never expire. This property is ignored if conserve-sockets is true.</p> <p>Valid values are in the range 0..600000.</p>	60000
ssl-ciphers	Used for SSL security. A space-separated list of the valid SSL ciphers for this connection. A setting of 'any' uses any ciphers that are enabled by default in the configured JSSE provider.	any
ssl-enabled	Used for SSL security. Boolean indicating whether to use SSL for member communications. A true setting requires the use of locators. This attribute must be consistent across the distributed system.	false
ssl-protocols	Used for SSL security. A space-separated list of the valid SSL protocols for this connection. A setting of 'any' uses any protocol that is enabled by default in the configured JSSE provider.	any
ssl-require-authentication	Used for SSL security. Boolean indicating whether to require authentication for member communication.	true
start-locator	<p>If set, automatically starts a locator in the current process when the member connects to the distributed system and stops the locator when the member disconnects.</p> <p>To use, specify the locator with an optional address or host specification and a required port number, in one of these formats:</p> <pre>start-locator=address[port1]</pre> <pre>start-locator=port1</pre> <p>If you only specify the port, the address assigned to the member is used for the locator. If not already there, this locator is automatically added to the list of locators in this set of <code>gemfire properties</code>.</p>	<i>not set</i>
statistic-archive-file	The file to which the running system member writes statistic samples. For example: "StatisticsArchiveFile.gfs". An empty string disables archiving. Adding .gz suffix to the file name causes it to be compressed.	<i>not set</i>
statistic-sample-rate	How often to sample statistics, in milliseconds.  Valid values are in the range 100..60000.	1000
statistic-sampling-enabled	<p>Whether to collect and archive statistics on the member.</p> <p>Statistics sampling provides valuable information for ongoing system tuning and troubleshooting purposes. Sampling statistics at the default sample rate does not impact system performance. We recommend enabling statistics sampling in production environments.</p> <p> <b>Note:</b> This setting does not apply to partitioned regions, where statistics are always enabled.</p>	false
tcp-port	<p>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.</p> <p>Valid values are in the range 0..65535.</p>	0
tombstone-gc-threshold	The number of tombstones that can accumulate before the GemFire member triggers garbage collection for tombstones. See <a href="#">How Destroy and Clear Operations Are Resolved</a> on page 243.	100000

Setting	Definition	Default
udp-fragment-size	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.  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 false, 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.  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
writable-working-dir	The writable directory where this member should persist runtime information about licensing. Keep this directory the same between runs for each member so the member application can read what it persisted on the last run.   <b>Note:</b> This property is deprecated as of 6.6.1. Use <code>license-working-dir</code> instead.	member's current working directory as determined by <code>System.getProperty("userdir")</code>

## Using Non-ASCII Strings in vFabric GemFire Property Files

You can specify Unicode (non-ASCII) characters in vFabric GemFire property files by using a \uXXXX escape sequence.

For a supplementary character, you need two escape sequences, one for each of the two UTF-16 code units. The XXXX denotes the 4 hexadecimal digits for the value of the UTF-16 code unit. For example, a properties file might have the following entries:

```
s1=hello there
s2=\u3053\u3093\u306b\u3061\u306f
```

For example, in `gemfire.properties`, you might write:

```
log-file=my\u00df.log
```

to indicate the desired property definition of `log-file=myß.log`.

If you have edited and saved the file in a non-ASCII encoding, you can convert it to ASCII with the `native2ascii` tool included in your Oracle Java distribution. For example, you might want to do this when editing a properties file in Shift\_JIS, a popular Japanese encoding.

For more information on internationalization in Java, see

<http://www.oracle.com/us/technologies/java/faq-jsp-138165.html>.

# Chapter 65

---

## cache.xml

Use the cache.xml file to set up general cache facilities and behavior and to create and initialize cached data regions. These sections document cache.xml requirements; provide hierarchical diagrams of <cache> and <client-cache> elements; and describe the function of each element.

### cache.xml Quick Reference

This section documents cache.xml file requirements and variables. It also points you to specific element sections for server, client, and WAN configuration.

- [Cache XML Requirements](#) on page 861
- [Variables in cache.xml](#) on page 861
- [Configuration Quick Reference](#) on page 862

### Cache XML Requirements

The cache.xml file has these requirements:

- The contents must conform to the XML definition in dtd/cache7\_0.dtd of your GemFire installation.
- The file must include a DOCTYPE of one of the following forms:
  - Server or peer cache:

```
<!DOCTYPE cache PUBLIC  
"-//GemStone Systems, Inc.//GemFire Declarative Caching 7.0//EN"  
"http://www.gemstone.com/dtd/cache7_0.dtd">
```

- Client cache:

```
<!DOCTYPE client-cache PUBLIC  
"-//GemStone Systems, Inc.//GemFire Declarative Caching 7.0//EN"  
"http://www.gemstone.com/dtd/cache7_0.dtd">
```

- Any class name specified in the file must have a public zero-argument constructor and must implement the com.gemstone.gemfire.cache.Declarable interface. Parameters declared in the XML for the class are passed to the class init method.

### Variables in cache.xml

You can use variables in the cache.xml to customize your settings without modifying the XML file.

Set your variables in Java system properties when you start your cache server or application process.

Example cache.xml with variables and the gfsh start server command that sets the variables:

```
<!DOCTYPE cache PUBLIC
  "-//GemStone Systems, Inc.//GemFire Declarative Caching 7.0//EN"
  "http://www.gemstone.com/dtd/cache7_0.dtd">
<cache>
  <cache-server port="${PORT}" max-connections="${MAXCNXS}" />
  <region name="root">
    <region-attributes refid="REPLICATE" />
  </region>
</cache>
```

```
gfsh>start server --name=server2 --cache-xml-file=cache.xml --J=-DPORT=30333
--J=-DMAXCNXS=77
```

## Configuration Quick Reference

To configure cache servers, clients, and WAN topologies, see the following sections:

- Server Configuration

- [<cache> Element Reference](#) on page 865
- [<cache-server>](#) on page 874
- [<region>](#) on page 905
- [<region-attributes>](#) on page 882

You can set the same server configuration properties using the `com.gemstone.gemfire.cache.server.CacheServer` and `com.gemstone.gemfire.cache.Cache` interfaces. For detailed information, see the online Java API documentation.

- Client Configuration

- [<client-cache> Element Reference](#) on page 915
- [<pool>](#) on page 917
- [<region>](#) on page 944

You can set the same client configuration properties using the `com.gemstone.gemfire.cache.client.ClientCache` and `Pool` interfaces. For detailed information, see the online Java API documentation.

- Multi-site (WAN) Configuration and Asynchronous Event Queue Configuration

- [<gateway-sender>](#) on page 868
- [<gateway-receiver>](#) on page 870
- [<async-event-queue>](#) on page 872

The gateway sender and receiver APIs in `com.gemstone.gemfire.cache.util` provide corresponding getter and setter methods for these attributes.

## <cache> Element Hierarchy

This section shows the hierarchy of `<cache>` element sub-elements that you use to configure GemFire caches and servers.

For details, click on one of the elements.

```
<cache> Element Reference on page 865
<cache-transaction-manager> on page 866
```

```
<transaction-listener> on page 867
<transaction-writer> on page 867
<dynamic-region-factory> on page 867
    <disk-dir>
<gateway-hub> on page 868
    <gateway>
        <gateway-endpoint>
        <gateway-listener>
        <gateway-queue>
<gateway-sender> on page 868
    <gateway-event-filter> on page 870
    <gateway-transport-filter> on page 871
<gateway-receiver> on page 870
    <gateway-transport-filter>

<gateway-conflict-resolver> on page 872
<async-event-queue> on page 872
    <async-event-listener> on page 873
        <class-name>
        <parameter>
            <string>
            <declarable>
<cache-server> on page 874
    <group> on page 875
    <client-subscription> on page 875
    <custom-load-probe> on page 876
        <class-name>
        <parameter>
            <string>
            <declarable>
<bridge-server> on page 876
<pool> on page 876
    <locator> on page 878
    <server> on page 879
<disk-store> on page 880
    <disk-dirs> on page 880
    <disk-dir> on page 880
<pdx> on page 881
    <pdx-serializer> on page 881
        <class-name>
        <parameter>
            <string>
            <declarable>
<region-attributes> on page 882
    <key-constraint> on page 889
    <value-constraint> on page 889
    <region-time-to-live> on page 890
        <expiration-attributes> on page 890
        <custom-expiry> on page 891
            <class-name>
            <parameter>
                <string>
                <declarable>
<region-idle-time> on page 891
    <expiration-attributes> on page 891
        <custom-expiry>
            <class-name>
            <parameter>
                <string>
                <declarable>
<entry-time-to-live> on page 893
```

```
<expiration-attributes>

    <custom-expiry>
        <class-name>
        <parameter>
            <string>
            <declarable>
<entry-idle-time> on page 894
    <expiration-attributes> on page 894
        <custom-expiry>
            <class-name>
            <parameter>
                <string>
                <declarable>
<disk-write-attributes> on page 896
    <asynchronous-writes> on page 896
    <synchronous-writes> on page 896
<disk-dirs> on page 880
    <disk-dir> on page 880
<partition-attributes> on page 896
    <partition-resolver> on page 897
        <class-name>
        <parameter>
            <string>
            <declarable>
    <partition-listener> on page 897
        <class-name>
        <parameter>
            <string>
            <declarable>
        <fixed-partition-attributes> on page 898
    <membership-attributes> on page 898
        <required-role> on page 899
    <subscription-attributes> on page 899
    <cache-listener> on page 900
        <class-name>
        <parameter>
            <string>
            <declarable>
    <eviction-attributes> on page 901
        <lru-entry-count> on page 901
        <lru-heap-percentage> on page 901
            <class-name>
            <parameter>
                <string>
                <declarable>
    <lru-memory-size> on page 902
        <class-name>
        <parameter>
            <string>
            <declarable>
<jndi-bindings> on page 902
    <jndi-bindings> on page 902
        <config-property> on page 904
            <config-property-name> on page 904
            <config-property-type> on page 904
            <config-property-value> on page 904
<region> on page 905
    <region-attributes> on page 882
    <index> on page 905
        <functional> on page 906
```

```

<primary-key> on page 906
<entry> on page 907
<key> on page 907
  <string>
    <declarable>
<value> on page 908
  <string>
    <declarable>
<region>
<vm-root-region>
<function-service> on page 908
  <function> on page 909
    <class-name>
      <parameter>
        <string>
          <declarable>
<resource-manager> on page 909
<serialization-registration> on page 910
  <serializer> on page 910
    <class-name>
  <instantiator> on page 910
    <class-name?>
<backup> on page 910
<initializer> on page 911
  <class-name>
  <parameter>
    <string>
    <declarable>
</cache>

```

## <cache> Element Reference

This section documents the `cache.xml` sub-elements used for GemFire server configuration. All elements are sub-elements of the `<cache>` element.

For GemFire client configuration, see [<client-cache> Element Reference](#) on page 915.

**API:**`com.gemstone.gemfire.cache.CacheFactory`

**Table 52: <cache> Element Attributes**

Attribute	Description	Default
copy-on-read	Boolean indicating whether entry value retrieval methods return direct references to the entry value objects in the cache (false) or copies of the objects (true).	False
is-server	Boolean indicating whether this member is a cache server.	False
lock-timeout	The timeout, in seconds, for implicit object lock requests. This setting affects automatic locking only, and does not apply to manual locking. If a lock request does not return before the specified timeout period, it is cancelled and returns with a failure.	60
lock-lease	The timeout, in seconds, for implicit and explicit object lock leases. This affects both automatic locking and manual locking. Once a lock is obtained, it can remain in force for the lock lease time period before being automatically cleared by the system.	120

Attribute	Description	Default
message-sync-interval	Used for client subscription queue synchronization when this member acts as a server to clients and server redundancy is used. Sets the frequency (in seconds) at which the primary server sends messages to its secondary servers to remove queued events that have already been processed by the clients.	1
search-timeout	How many seconds a <code>netSearch</code> operation can wait for data before timing out. You may want to change this based on your knowledge of the network load or other factors.	300

**Example:**

```
<cache>
  <cache-server port="40404" />
  <region name="root">
    <region-attributes refid="REPLICATE" />
    <region name="cs_region" refid="REPLICATE" >
      <region-attributes>
        <cache-loader>
          <class-name>cacheRunner.StringLoader</class-name>
        </cache-loader>
        <cache-listener>
          <class-name>cacheRunner.LoggingCacheListener</class-name>
        </cache-listener>
      </region-attributes>
    </region>
  </region>
</cache>
```

**<cache-transaction-manager>**

Specifies a transaction listener.

**API:** CacheTransactionManager

**Example:**

```
<cache search-timeout="60">
  <cache-transaction-manager>
    <transaction-listener>
      <class-name>com.company.data.MyTransactionListener</class-name>
      <parameter name="URL">
        <string>jdbc:cloudscape:rmi:MyData</string>
      </parameter>
    </transaction-listener>
    <transaction-listener>... </transaction-listener>
    <transaction-writer>
      <class-name>com.company.data.MyTransactionWriter</class-name>
      <parameter name="URL">
        <string>jdbc:cloudscape:rmi:MyData</string>
      </parameter>
      <parameter
    </transaction-writer>
```

```
</cache-transaction-manager> . . .
</cache>
```

### **<transaction-listener>**

When a transaction ends, its thread calls the TransactionListener to perform the appropriate follow-up for successful commits, failed commits, or voluntary rollbacks.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name> and <parameter>](#) on page 911.

### **<transaction-writer>**

When you commit a transaction, a TransactionWriter can perform additional tasks, including aborting the transaction.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name> and <parameter>](#) on page 911.

### **<dynamic-region-factory>**

The <dynamic-region-factory> element configures a dynamic region factory for this cache. You can use this element to dynamically create regions in your application code. Use the `createDynamicRegion()` method of the `com.gemstone.gemfire.cache.DynamicRegionFactory` class in your Java code to dynamically create regions.



**Note:** You can not use this element to dynamically create *partitioned* regions.

Pivotal Inc. recommends that you use functions to dynamically create regions. See [Creating Regions Dynamically](#) on page 208.

The optional <disk-dir> sub-element specifies the directory to store the persistent files that are used for dynamic region bookkeeping. It defaults to the current directory.

Set the pool-name attribute to set the name of the connection pool used by client applications in a client/server cache configuration. Do not specify the pool-name attribute in servers or peers.

**API:** `com.gemstone.gemfire.cache.DynamicRegionFactory`

**Table 53: Attributes**

Attribute	Description	Default
disable-persist-backup	When set to <code>false</code> , the factory is persisted on disk.	<code>false</code>
disable-register-interest	When set to <code>false</code> , client regions created by the factory register interest in all keys in a corresponding server cache region.	<code>false</code>
pool-name	The name of a connection pool used by the client factory to communicate with the server-side factory.	<code>none</code>

#### **Example:**

```
<dynamic-region-factory
  pool-name=myPool>
  <disk-dir>/home/gemfire/myDiskdir</disk-dir>
</dynamic-region-factory>
```

### **<disk-dir>**

Specifies a region or disk store's disk directory.

**Table 54: Attributes**

Attribute	Description	Default
dir-size	Maximum amount of space to use for the disk store, in megabytes.	214748364 (2 petabytes)

**Example:**

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

**<gateway-hub>**

Deprecated See [<gateway-sender>](#) on page 868 and [<gateway-receiver>](#) on page 870 instead.

**<gateway>**

Deprecated

**<gateway-endpoint>**

Deprecated

**<gateway-listener>**

Deprecated

**<gateway-queue>**

Deprecated

**<gateway-sender>**

Configures a gateway sender to distribute region events to another GemFire site. See [Configuring a Multi-site \(WAN\) System](#) on page 195.

**API:** `GatewaySender`

**Table 55: Attributes**

Attribute	Description	Default						
parallel	Value of "true" or "false" that specifies the type of gateway sender that GemFire creates.	false						
dispatcher-threads	Number of dispatcher threads that are used to process region events from a serial gateway sender queue.	1						
order-policy	<p>When the <code>dispatcher-threads</code> attribute is greater than 1, <code>order-policy</code> configures the way in which multiple dispatcher threads process region events from the gateway queue. This attribute can have one of the following values:</p> <table> <tr> <td>key</td> <td>When distributing region events from the local queue, multiple dispatcher threads preserve the order of key updates.</td> </tr> <tr> <td>thread</td> <td>When distributing region events from the local queue, multiple dispatcher threads preserve the order in which a given thread added region events to the queue.</td> </tr> <tr> <td>partition</td> <td>When distributing region events from the local queue, multiple dispatcher threads preserve the order in which region events were added to the local queue. For a partitioned region, this means that all region events delivered to a specific partition are delivered in the same order to the remote GemFire site. For a distributed region, this means that all key updates delivered to the local gateway sender queue are distributed to the remote site in the same order.</td> </tr> </table>	key	When distributing region events from the local queue, multiple dispatcher threads preserve the order of key updates.	thread	When distributing region events from the local queue, multiple dispatcher threads preserve the order in which a given thread added region events to the queue.	partition	When distributing region events from the local queue, multiple dispatcher threads preserve the order in which region events were added to the local queue. For a partitioned region, this means that all region events delivered to a specific partition are delivered in the same order to the remote GemFire site. For a distributed region, this means that all key updates delivered to the local gateway sender queue are distributed to the remote site in the same order.	key
key	When distributing region events from the local queue, multiple dispatcher threads preserve the order of key updates.							
thread	When distributing region events from the local queue, multiple dispatcher threads preserve the order in which a given thread added region events to the queue.							
partition	When distributing region events from the local queue, multiple dispatcher threads preserve the order in which region events were added to the local queue. For a partitioned region, this means that all region events delivered to a specific partition are delivered in the same order to the remote GemFire site. For a distributed region, this means that all key updates delivered to the local gateway sender queue are distributed to the remote site in the same order.							

Attribute	Description	Default
id	Unique identifier for the gateway sender, usually an identifier associated with a physical location. This attribute is required.	null
remote-distributed-system-id	Integer that uniquely identifies the remote GemFire cluster to which this gateway sender will send region events. This value corresponds to the <code>distributed-system-id</code> property specified in locators for the remote cluster. This attribute is required.	null
manual-start	Boolean value that specifies whether you need to manually start the gateway sender. If you supply a null value, the default is "false" and the gateway sender attempts to start automatically.	null (false)
socket-buffer-size	Size of the socket buffer that sends messages to remote sites. This size should match the size of the <code>socket-buffer-size</code> attribute of remote gateway receivers that process region events.	32765
socket-read-timeout	Amount of time in milliseconds that this gateway sender blocks while waiting to receive an acknowledgment from a remote site.	10000 milliseconds
enable-batch-conflation	Boolean value that determines whether GemFire should conflate messages.	false
batch-size	Maximum number of messages that a batch can contain.	100
batch-time-interval	Maximum number of milliseconds that can elapse between sending batches.	1000
enable-persistence	Boolean value that determines whether GemFire persists the gateway queue.	false
disk-store-name	Named disk store to use for storing the queue overflow, or for persisting the queue. If you specify a value, the named disk store must exist. If you specify a null value, GemFire uses the default disk store for overflow and queue persistence.	
disk-synchronous	For regions that write to disk, boolean that specifies whether disk writes are done synchronously for the region.	true
maximum-queue-memory	Maximum amount of memory in megabytes that the queue can consume before overflowing to disk.	100 MB
alert-threshold	Maximum number of milliseconds that a region event can remain in the gateway sender queue before GemFire logs an alert.	0

**Example:**

```
<cache>

  <gateway-sender
    id="remoteA"
    parallel="true"
    remote-distributed-system-id="1">
    <gateway-event-filter>
      <class-name>com.gemstone.gemfire.util.SampleEventFilter</class-name>

      <parameter
        name="param1">
        <string>"value1"</string>
      </parameter>
    </gateway-event-filter>
    <gateway-transport-filter>

      <class-name>com.gemstone.gemfire.util.SampleTransportFilter</class-name>
      <parameter
        name="param1">
        <string>"value1"</string>
      </parameter>
    </gateway-transport-filter>
  </gateway-sender>
```

```
...
</cache>
```

### **<gateway-event-filter>**

A GatewayEventFilter implementation determines whether a region event is placed in a gateway sender queue and/or whether an event in a gateway queue is distributed to a remote site. You can optionally add one or more GatewayEventFilter implementations to a gateway sender, either in the cache.xml configuration file or using the Java API.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name> and <parameter>](#) on page 911.

#### **Example:**

```
<gateway-event-filter>
    <class-name>com.gemstone.gemfire.util.SampleEventFilter</class-name>

    <parameter name="param1">
        <string>"value1"</string>
    </parameter>
</gateway-event-filter>
```

### **<gateway-transport-filter>**

Use a GatewayTransportFilter implementation to process the TCP stream that sends a batch of events that is distributed from one GemFire cluster to another over a WAN. A GatewayTransportFilter is typically used to perform encryption or compression on the data that is distributed. You install the same GatewayTransportFilter implementation on both a gateway sender and gateway receiver.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name> and <parameter>](#) on page 911.

#### **Example:**

```
<gateway-transport-filter>

<class-name>com.gemstone.gemfire.util.SampleTransportFilter</class-name>
    <parameter
        name="param1">
        <string>"value1"</string>
    </parameter>
</gateway-transport-filter>
```

### **<gateway-receiver>**

Configures a gateway receiver to receive and apply region events that were distributed from another GemFire site. See [Configuring a Multi-site \(WAN\) System](#) on page 195.

**API:** `GatewayEventFilter`, `GatewayTransportFilter`

**Table 56: Attributes**

Attribute	Description	Default
start-port	Starting port number to use when specifying the range of possible port numbers this gateway receiver will use to connect to gateway senders in other sites. GemFire chooses an unused port number in the specified port number range to start the receiver. If no port numbers in the range are available, an exception is thrown.	5000

Attribute	Description	Default
	The STARTPORT value is inclusive while the ENDPORT value is exclusive. For example, if you specify STARTPORT= "50510" and ENDPOINT= "50520", GemFire chooses a port value from 50510 to 50519.	
end-port	Defines the upper bound port number to use when specifying the range of possible port numbers this gateway receiver will use to for connections from gateway senders in other sites. GemFire chooses an unused port number in the specified port number range to start the receiver. If no port numbers in the range are available, an exception is thrown.  The ENDPORT value is exclusive while the STARTPORT value is inclusive. For example, if you specify STARTPORT= "50510" and ENDPOINT= "50520", GemFire chooses a port value from 50510 to 50519.	5500
bind-address	Network address for connections from gateway senders in other sites. Specify the address as a literal string value.	
maximum-time-between-pings	Integer value that specifies the time interval (in milliseconds) to use between pings to connected WAN sites. This value determines the maximum amount of time that can elapse before a remote WAN site is considered offline.	60000
socket-buffer-size	An integer value that sets the buffer size (in bytes) of the socket connection for this gateway receiver. This value should match the <code>socket-buffer-size</code> setting of gateway senders that connect to this receiver.	32768

**Example:**

```
<cache>
    <gateway-receiver start-port="1530" end-port="1551">
        <gateway-transport-filter>>

<class-name>com.gemstone.gemfire.util.SampleTransportFilter</class-name>
    <parameter
        name="param1">
        <string>"value1"</string>
    </parameter>
</gateway-transport-filter>
</gateway-receiver>
</cache>
```

**<gateway-transport-filter>**

Use a `GatewayTransportFilter` implementation to process the TCP stream that sends a batch of events that is distributed from one GemFire cluster to another over a WAN. A `GatewayTransportFilter` is typically used to perform encryption or compression on the data that distributed. You install the same `GatewayTransportFilter` implementation on both a gateway sender and gateway receiver.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name> and <parameter>](#) on page 911.

**Example:**

```
<gateway-transport-filter>

<class-name>com.gemstone.gemfire.util.SampleTransportFilter</class-name>
    <parameter
        name="param1">
        <string>"value1"</string>
```

```
</parameter>
</gateway-transport-filter>
```

### **<gateway-conflict-resolver>**

An event-handler plug-in that is called in order to determine whether a potentially conflicting WAN update should be applied to the local cache. A `GatewayConflictResolver` is invoked if the current value in a cache entry was established by a different distributed system (with a different distributed-system-id) than an event that is attempting to modify the entry. It is not invoked if the event has the same distributed system ID as the event that last changed the entry. See [Resolving Conflicting Events](#) on page 200

Specify the Java class for the gateway conflict resolver plug-in and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [`<class-name>` and `<parameter>`](#) on page 911.

**API:** `com.gemstone.gemfire.cache.util.GatewayConflictResolver`

**Example:**

```
<gateway-conflict-resolver>
  <class-name>
    myPackage.MyConflictResolver
  </class-name>
</gateway-conflict-resolver>
```

### **<async-event-queue>**

Configures a queue for sending region events to an `AsyncEventListener` implementation (for example, for write-behind event handling). See [`<gateway-sender>`](#).

**API:** `com.gemstone.gemfire.cache.asyncqueue.AsyncEventQueue`

**Table 57: Attributes of `<async-event-queue>`**

Attribute	Description	Default
<code>id</code>	Unique identifier for the queue. This attribute is required.	null
<code>parallel</code>	Value of "true" or "false" that specifies the type of queue that GemFire creates.	false
<code>batch-size</code>	Maximum number of messages that a batch can contain.	100
<code>batch-time-interval</code>	Maximum number of milliseconds that can elapse between sending batches.	1000
<code>enable-batch-conflation</code>	Boolean value that determines whether GemFire should conflate messages.	false
<code>disk-store-name</code>	Named disk store to use for storing queue overflow, or for persisting the queue. If you specify a value, the named disk store must exist. If you specify a null value, GemFire uses the default disk store for overflow and queue persistence.	null specifies the default disk store
<code>disk-synchronous</code>	For regions that write to disk, boolean that specifies whether disk writes are done synchronously for the region.	true
<code>dispatcher-threads</code>	Number of dispatcher threads that are used to process region events from the queue.	1
<code>maximum-queue-memory</code>	Maximum amount of memory in megabytes that the queue can consume before overflowing to disk.	100 mb
<code>order-policy</code>	When the <code>dispatcher-threads</code> attribute is greater than 1, <code>order-policy</code> configures the way in which multiple dispatcher threads process region events from the queue. This attribute can have one of the following values: <ul style="list-style-type: none"> <li>• <b>key</b>. When distributing region events from the local queue, multiple dispatcher threads preserve the order of key updates.</li> <li>• <b>thread</b>. When distributing region events from the local queue, multiple dispatcher threads preserve the order in which a given thread added region events to the queue.</li> </ul>	key

Attribute	Description	Default
	<ul style="list-style-type: none"> <li><b>partition.</b> This option is valid for parallel event queues. When distributing region events from the local queue, multiple dispatcher threads preserve the order in which region events were added to the local queue. For a partitioned region, this means that all region events delivered to a specific partition are delivered in the same order to the remote GemFire site. For a distributed region, this means that all key updates delivered to the local queue are distributed to the remote site in the same order.</li> </ul>	
persistent	Boolean value that determines whether GemFire persists this queue.	False

**Example:**

```
<cache>
  <async-event-queue
    id="sampleQueue"
    persistent="true"
    disk-store-name="exampleStore"
    parallel="false">
    <async-event-listener>
      <class-name>MyAsyncEventListener</class-name>
      <parameter name="url">
        <string>jdbc:db2:SAMPLE</string>
      </parameter>
      <parameter name="username">
        <string>gfeadmin</string>
      </parameter>
      <parameter name="password">
        <string>admin1</string>
      </parameter>
    </async-event-listener>
  </async-event-queue>
  ...
</cache>
```

**<async-event-listener>**

An AsyncEventListener receives callbacks for events that change region data. You can use an AsyncEventListener implementation as a write-behind cache event handler to synchronize region updates with a database.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name> and <parameter>](#) on page 911.

**API:** com.gemstone.gemfire.cache.asyncqueue.AsyncEventListener

**Example:**

```
...
  <async-event-listener>
    <class-name>MyAsyncEventListener</class-name>
    <parameter name="url">
      <string>jdbc:db2:SAMPLE</string>
    </parameter>
    <parameter name="username">
      <string>gfeadmin</string>
    </parameter>
    <parameter name="password">
      <string>admin1</string>
    </parameter>
```

```
</async-event-listener>
```

```
...
```

### **<cache-server>**

Configures the cache to serve region data to clients in a client/server caching system. This element indicates the port the server listens on for client communication.

The `cacheserver` process uses only `cache.xml` configuration. For application servers, you can set the same configuration properties using the `com.gemstone.gemfire.cache.server.CacheServer` and `com.gemstone.gemfire.cache.Cache` interfaces. For detailed information, see the online Java API documentation.

**API:** `com.gemstone.gemfire.cache.server.CacheServer`

**Table 58: Attributes of <cache-server>**

Attribute	Description	Default Value
bind-address	Hostname or IP address that the server is to listen on for client connections. If null, the server listens on the machine's default address.	null
hostname-for-clients	Hostname or IP address to pass to the client as the location where the server is listening. When the server connects to the locator it tells the locator the host and port where it is listening for client connections. If the host the server uses by default is one that the client can't translate into an IP address, the client will have no route to the server's host and won't be able to find the server. For this situation, you must supply the server's alternate hostname for the locator to pass to the client. If null, the server's <code>bind-address</code> setting is used.	null
load-poll-interval	Frequency, in milliseconds, to poll the load probe for load information on the server.	5000 (5 seconds)
max-connections	Maximum number of client connections for the server. When the maximum is reached, the server refuses additional client connections.   <b>Note:</b> Set this at least as high as <code>max-threads</code> .	800
max-threads	Maximum number of threads allowed in this server to service client connections. When the limit is reached, server threads begin servicing multiple connections. A zero setting causes the server to use a thread for every client connection.   <b>Note:</b> Set this no higher than <code>max-connections</code> .	0
maximum-message-count	Maximum number of messages allowed in a subscription queue. When the queue reaches this limit, messages block.   <b>Note:</b> Used only if <code>client-subscription</code> is not configured.	230000
maximum-time-between-pings	Maximum time, in milliseconds, the server allows to pass between messages or pings indicating a client is healthy.   <b>Note:</b> A setting of 0 or a negative number turns off client health monitoring. Be careful not to do this accidentally.	60000 (1 minute)

Attribute	Description	Default Value
message-time-to-live	Setting used for highly available subscription queues. The expiration time, in seconds, for non-durable messages in the secondary server's client subscription queue. The system removes non-durable messages that have been in the queue beyond this time. If set to 0 (zero), the messages are never removed.   <b>Note:</b> Set this high enough to avoid removing messages that are still valid, to avoid losing messages during server failover.	180 (3 minutes)
port	Port that the server listens on for client communication.	40404
socket-buffer-size	Size for socket buffers used for server-to-client communication.	32768

**Example:**

```
<cache>
  <cache-server
    port="40404"
  />
  ...
</cache>
```

**<group>**

Deprecated as of GemFire 7.0. Specify server groups in the gemfire.properties file. See [Organizing Servers Into Logical Member Groups](#) on page 184.

**<group>**

Deprecated as of GemFire 7.0. Specify server groups in the gemfire.properties file. See [Organizing Servers Into Logical Member Groups](#) on page 184.

**<client-subscription>**

Overflow specification for client subscription queues. Sets a capacity limit on the in-memory queue and specifies where to overflow when capacity is reached. By default no overflow is used. Specified in three parts:

**Default:** no overflow

**API:****Table 59: Attributes**

Attribute	Description	Default
eviction-policy	How the capacity is calculated. The options are mem for memory use, entry for message count, and null for no overflow.	null
capacity	. Used if eviction-policy is not none. Specified in megabytes for mem and as a positive integer for entry	1
disk-store-name	Used if eviction-policy is not none. Default: default disk store. If specified, the disk-store-name must specify a disk store that is already defined in the cache.	default disk store
overflow-directory	DEPRECATED as of GemFire version 6.5. See the disk-store-name attribute of the element <a href="#">&lt;region-attributes&gt;</a> on page 882.	

**Example:**

```
<cache-server port=4444>
</cache>
```

**<client-subscription>**

Overflow specification for client subscription queues. Sets a capacity limit on the in-memory queue and specifies where to overflow when capacity is reached. By default no overflow is used. Specified in three parts:

- eviction-policy
- capacity
- disk-store-name

**Default:** no overflow

**API:** com.gemstone.gemfire.cache.server.CacheServer.getClientSubscriptionConfig

**Table 60: Attributes**

Attribute	Description	Default
eviction-policy	How the capacity is calculated. The options are mem for memory use, entry for message count, and null for no overflow.	null
capacity	Used if eviction-policy is not none. Specified in megabytes for mem and as a positive integer for entry.	1
disk-store-name	Used if eviction-policy is not none. Default: default disk store. If specified, the disk-store-name must specify a disk store that is already defined in the cache.	default disk store
overflow-directory	DEPRECATED as of GemFire version 6.5. See the <code>disk-store-name</code> attribute of the element <a href="#">&lt;region-attributes&gt;</a> on page 882.	

**Example:**

```
<cache-server port=4444>
  <client-subscription
    eviction-policy="entry"
    capacity=35
    disk-store-name="OverflowDir">
  </client-subscription>
</cache-server>
```

### **<custom-load-probe>**

Application plug-in used to provide current and predicted server load information to the server locators.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name> and <parameter>](#) on page 911.

**Default:** If this is not defined, the default GemFire load probe is used.

**API:** com.gemstone.gemfire.cache.server.setLoadProbe

**Example:**

```
<custom-load-probe>
  <class-name>
    myPackage.MyLoadProbe
  </class-name>
</custom-load-probe>
```

### **<bridge-server>**

DEPRECATED as of GemFire version 5.7. See [<cache-server>](#) on page 874.

### **<pool>**

Use for client caches. Defines a client's server pool used to communicate with servers running in a different distributed system.

**API:** com.gemstone.gemfire.cache.client.PoolFactory

**Table 61: Attributes**

Attribute	Description	Default
free-connection-timeout	Amount of time a thread will wait to get a pool connection before timing out with an exception. This timeout keeps threads from waiting indefinitely when the pool's <b>max-connections</b> has been reached and all connections in the pool are in use by other threads.	10000
idle-timeout	Maximum time, in milliseconds, a pool connection can stay open without being used when there are more than <b>min-connections</b> in the pool. Pings over the connection do not count as connection use. If set to -1, there is no idle timeout.	5000
load-conditioning-interval	Amount of time, in milliseconds, a pool connection can remain open before being eligible for silent replacement to a less-loaded server.	300000 (5 minutes)
max-connections	Maximum number of pool connections the pool can create. If the maximum connections are in use, an operation requiring a client-to-server connection blocks until a connection becomes available or the <b>free-connection-timeout</b> is reached. If set to -1, there is no maximum. The setting must indicate a cap greater than <b>min-connections</b> .   <b>Note:</b> If you need to use this to cap your pool connections, you should disable the pool attribute <b>pr-single-hop-enabled</b> . Leaving single hop enabled can increase thrashing and lower performance.	-1
min-connections	Minimum number of pool connections to keep available at all times. Used to establish the initial connection pool. If set to 0 (zero), no connection is created until an operation requires it. This number is the starting point, with more connections added later as needed, up to the <b>max-connection</b> setting. The setting must be an integer greater than or equal to 0.	1
multiuser-authentication	Used for installations with security where you want to accommodate multiple users within a single client. If set to true, the pool provides authorization for multiple user instances in the same client application, and each user accesses the cache through its own <b>RegionService</b> instance. If false, the client either uses no authorization or just provides credentials for the single client process.	false
name	Name of this pool. Used in the client region pool-name to assign this pool to a region in the client cache.   <b>Note:</b> This is a required property with no default setting.	none
ping-interval	How often to communicate with the server to show the client is alive, set in milliseconds. Pings are only sent when the ping-interval elapses between normal client messages.   <b>Note:</b> Set this lower than the server's <b>maximum-time-between-pings</b> .	10000
pr-single-hop-enabled	Setting used to improve access to partitioned region data in the servers. Indicates whether to use metadata about the partitioned region data storage locations to decide where to send some data requests. This allows a client to send a data operation directly to the server hosting the key. Without this, the client contacts any available server and that server contacts the data store. This is used only for operations that can be carried out on a server-by-server basis, like put, get, and destroy.	true
read-timeout	Maximum time, in milliseconds, for the client to wait for a response from a server.	10000
retry-attempts	Number of times to retry a client request before giving up. If one server fails, the pool moves to the next, and so on until it is successful or it hits this limit. If the available servers are fewer than this setting, the pool will retry servers that have	-1

Attribute	Description	Default
	already failed until it reaches the limit. If this is set to -1, the pool tries every available server once.	
server-group	Logical named server group to use from the pool. A null value uses the global server group to which all servers belong.   <b>Note:</b> This is only used when the <code>locator</code> list is defined.	null
socket-buffer-size	Size for socket buffers from the client to the server. Default: 32768.	32768
statistic-interval	Interval, in milliseconds, at which to send client statistics to the server. If set to -1, statistics are not sent.	-1
subscription-ack-interval	Time, in milliseconds, between messages to the primary server to acknowledge event receipt.   <b>Note:</b> Used only when <code>subscription-redundancy</code> is not '0' (zero).	500
subscription-enabled	Boolean indicating whether the server should connect back to the client and automatically sends server-side cache update information. Any bind address information for the client is automatically passed to the server for use in the callbacks.	false
subscription-message-tracking-timeout	Time-to-live, in milliseconds, for entries in the client's message tracking list.	900000 (15 minutes)
subscription-redundancy	Number of servers to use as backup to the primary for highly available subscription queue management. If set to 0, none are used. If set to -1, all available servers are used.	0
thread-local-connections	Boolean specifying whether connections are sticky. True causes the connection to stick to the thread for multiple requests. False causes each connection to be returned to the pool after a request finishes. A sticky connection is returned to the pool when the thread releases it through the <code>Pool</code> method <code>releaseThreadLocalConnection</code> , when the <code>idle-timeout</code> is reached, or when the pool is destroyed.	false

**Example:**

```
<pool
    name="publisher"
    subscription-enabled="true">
    <locator
        host="myLocatorAddress1"
        port="12345"/>
    <locator
        host="myLocatorAddress2"
        port="45678"/>
</pool>
```

**<locator>**

Addresses and ports of the locators to connect to. You can define multiple locators for the pool.



**Note:** Provide a locator list or `server` list, but not both.

**API:** `com.gemstone.gemfire.distributed.LocatorLauncher`

**Table 62: Attributes**

Attribute	Description	Default
host	Hostname of the locator	
port	Port number of the locator	

**Example:**

```
<pool ...>
<locator
    host="myLocatorHost"
    port="12345" />
```

**<server>**

Addresses and ports of the servers to connect to.

 **Note:** Provide a server list or locator list, but not both.
**Default:****API:****Table 63: Attributes**

Attribute	Description	Default
host	Hostname of the server	
port	Port number of the server	

**Example:**

```
<server
</pool>
```

**<server>**

Addresses and ports of the servers to connect to.

 **Note:** Provide a server list or locator list, but not both.
**Default:****API:** com.gemstone.gemfire.distributed.ServerLauncher**Table 64: Attributes**

Attribute	Description	Default
host	Hostname of the server	
port	Port number of the server	

**Example:**

```
<pool ...>
  <server
    host="myServerHost"
    port="123456" />
</pool>
```

**<disk-store>**

Defines a pool of one or more disk stores, which can be used by regions, client subscription queues, and gateway sender queues for WAN distribution.

**Default:** The cache default disk store, named "DEFAULT", is used when disk is used but no disk store is named.

**API:** com.gemstone.gemfire.cache.DiskStore

**Table 65: Attributes**

Attribute	Description	Default
name	The name of the Disk Store.	
auto-compact	Set to true to automatically compact the disk files.	
compaction-threshold	The threshold at which an oplog will become compactable. Until it reaches this threshold the oplog will not be compacted.  The threshold is a percentage in the range 0 to 100.	
allow-force-compaction	Set to true to allow disk compaction to be forced on this disk store.	
max-oplog-size	The maximum size, in megabytes, of an oplog (operation log) file.	
time-interval	The number of milliseconds that can elapse before unwritten data is written to disk.	
write-buffer-size	The size of the write buffer that this disk store uses when writing data to disk. Larger values may increase performance but use more memory. The disk store allocates one direct memory buffer of this size.	
queue-size	Maximum number of operations that can be asynchronously queued to be written to disk.	

**Example:**

```
<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>
```

**<disk-dirs>**

DEPRECATED as of GemFire version 6.5. See the `disk-store-name` attribute of the [<region-attributes>](#) on page 882 element.

**<disk-dir>**

Specifies a region or disk store's disk directory.

**Table 66: Attributes**

Attribute	Description	Default
dir-size	Maximum amount of space to use for the disk store, in megabytes.	214748364 (2 petabytes)

**Example:**

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

**<pdx>**

Specifies the configuration for the Portable Data eXchange (PDX) method of serialization.

**API:** com.gemstone.gemfire.cache.CacheFactory.setPdxReadSerialized, setPdxDiskStore, setPdxPersistent, setPdxIgnoreUnreadFields and com.gemstone.gemfire.cache.ClientCacheFactory.setPdxReadSerialized, setPdxDiskStore, setPdxPersistent, setPdxIgnoreUnreadFields

**Table 67: Attributes**

Attribute	Description	Default
read-serialized	Set it to true if you want PDX deserialization to produce a PdxInstance instead of an instance of the domain class.	
ignore-unread-fields	Set it to true if you do not want unread PDX fields to be preserved during deserialization. You can use this option to save memory. Set to true only in members that are only reading data from the cache.	
persistent	Set to true if you are using persistent regions or WAN gateways. This causes the PDX type information to be written to disk.	
disk-store-name	If using persistence, this attribute allows you to configure the disk store that the PDX type data will be stored in. By default, the default disk store is used.	

**Example:**

```
<cache>
  <pdx persistent="true" disk-store-name="myDiskStore">
    <pdx-serializer>
      <class-name>
        com.gemstone.gemfire.pdx.ReflectionBasedAutoSerializer
      </class-name>
      <parameter name="classes">
        <string>com.company.domain.DomainObject</string>
      </parameter>
    </pdx-serializer>
  </pdx>
  ...
</cache>
```

**<pdx-serializer>**

Allows you to configure the PdxSerializer for this GemFire member.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**Default:**

**API:** PdxSerializer

**Example:**

```
<cache>
```

**<pdx-serializer>**

Allows you to configure the PdxSerializer for this GemFire member.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**Default:**

**API:** com.gemstone.gemfire.cache.CacheFactory.setPdxSerializer

**Example:**

```
<cache>
  <pdx>
    <pdx-serializer>
      <class-name>com.company.ExamplePdxSerializer</class-name>
    </pdx-serializer>
  </pdx>
  ...
</cache>
```

**<region-attributes>**

Specifies a region attributes template that can be named (by id) and referenced (by refid) later in the cache.xml and through the API.

**API:** com.gemstone.gemfire.cache.RegionFactory or  
com.gemstone.gemfire.cache.ClientRegionFactory

**Table 68: Attributes**

Attribute	Description	Default
concurrency-level	<p>Gives an estimate of the maximum number of application threads that will concurrently access a region entry at one time. This attribute does not apply to partitioned regions. This attribute helps GemFire optimize the use of system resources and reduce thread contention. This sets an initial parameter on the underlying java.util.ConcurrentHashMap used for storing region entries.</p> <p> <b>Note:</b> Before you modify this, read the concurrency level description, then see the Java API documentation for java.util.ConcurrentHashMap.</p> <p><b>API:</b> setConcurrencyLevel</p> <p><b>Example:</b></p> <pre>&lt;region-attributes   concurrency-level="10"&gt; &lt;/region-attributes&gt;</pre>	16 (threads)
data-policy	<p>Specifies how the local cache handles data for a region. This setting controls behavior such as local data storage and region initialization.</p> <p> <b>Note:</b> Configure the most common options using the region shortcuts, RegionShortcut and ClientRegionShortcut. The default data-policy of normal specifies local cache storage. The empty policy specifies no local storage. In the region shortcuts, empty corresponds to the settings with the string PROXY. You can use an empty region for event delivery to and from the local cache without the memory overhead of data storage.</p> <p>You can specify the following data policies:</p>	normal

Attribute	Description	Default
	<p>empty      No data storage in the local cache. The region always appears empty. Use this for event delivery to and from the local cache without the memory overhead of data storage - zero-footprint producers that only distribute data to others and zero-footprint consumers that only see events. To receive events with this, set the region's <b>subscription-attributes interest-policy</b> to all.</p> <p>normal     Data used locally (accessed with <code>get</code>s, stored with <code>put</code>s, etc.) is stored in the local cache. This policy allows the contents in the cache to differ from other caches.</p> <p>partition   Data is partitioned across local and remote caches using the automatic data distribution behavior of partitioned regions. Additional configuration is done in the <b>partition-attributes</b>.</p> <p>replicate   The region is initialized with the data from other caches. After initialization, all events for the distributed region are automatically copied into the local region, maintaining a replica of the entire distributed region in the local cache. Operations that would cause the contents to differ with other caches are not allowed. This is compatible with local <b>SCOPE</b>, behaving the same as for normal.</p> <p>persistent-partition   Behaves the same as <b>partition</b> and also persists data to disk.</p> <p>persistent-replicate   Behaves the same as <b>replicate</b> and also persists data to disk.</p> <p>preloaded   Initializes like a replicated region, then, once initialized, behaves like a normal region.</p> <p><b>API:</b> <code>setDataPolicy</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     data-policy="replicate"&gt; &lt;/region-attributes&gt;</pre> <p>This is similar to using a region shortcut with <code>refid</code>, however when you use the <code>REPLICATE</code> region shortcut, it automatically sets the region's scope to <code>distributed-ack</code>.</p> <pre>&lt;region-attributes     refid="REPLICATE"&gt; &lt;/region-attributes&gt;</pre> <p>If you use <code>data-policy</code>, you must set the scope explicitly.</p>	
early-ack	Deprecated	
enable-async-conflation	<p>For TCP/IP distributions between peers, specifies whether to allow aggregation of asynchronous messages sent by the producer member for the region. This is a special-purpose boolean attribute that applies only when asynchronous queues are used for slow consumers. A false value disables conflation so that all asynchronous messages are sent individually. This special-purpose attribute gives you extra control over peer-to-peer communication between distributed regions using TCP/IP. This attribute does not apply to client/server communication, multi-site communication, or to communication using the UDP unicast or IP multicast protocols.</p> <p> <b>Note:</b> To use this attribute, the <code>multicast-enabled</code> region attribute <code>disable-tcp</code> in <code>gemfire.properties</code> must be false (the default for both). In addition, asynchronous queues must be enabled for slow consumers, specified with the <code>async*</code> gemfire properties.</p> <p><b>API:</b> <code>setEnableAsyncConflation</code></p>	true

Attribute	Description	Default
	<p><b>Example:</b></p> <pre>&lt;region-attributes     enable-async-conflation="false"&gt; &lt;/region-attributes&gt;</pre>	
enable-gateway	<p>Determines whether the gateway is enabled for the region. When set to true, events in the region are sent to the defined gateway hubs.</p> <p>Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the <code>gateway-sender-id</code> attribute of the <code>&lt;region-attributes&gt;</code> element.</p>	false
enable-subscription-conflation	<p>Boolean for server regions that specifies whether the server can conflate its messages to the client. A true value enables conflation.</p> <p> <b>Note:</b> The client can override this setting with the <code>conflate-events</code> property in its <code>gemfire.properties</code>.</p> <p><b>API:</b> <code>setEnableSubscriptionConflation</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     enable-subscription-conflation="true"&gt; &lt;/region-attributes&gt;</pre>	false
gateway-sender-ids	<p>Specifies one or more gateway sender IDs to use for distributing region events to remote GemFire sites. Specify multiple IDs as a comma-separated list.</p> <p><b>API:</b> <code>addGatewaySenderId</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     gateway-sender-ids="nwsender,swsender"&gt; &lt;/region-attributes&gt;</pre>	not set
async-event-queue-ids	<p>Specifies one or more asynchronous event queues to use for distributing region events an <code>AsyncEventListener</code> implementation (for example, for write-behind cache event handling). Specify multiple IDs as a comma-separated list.</p> <p><b>API:</b> <code>addAsyncEventQueueId</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     async-event-queue-ids="customerqueue,ordersqueue"&gt; &lt;/region-attributes&gt;</pre>	not set
hub-id	<p>If the <code>enable-gateway</code> attribute is set to true, a comma-separated list of gateway hub IDs that receive events from the region.</p> <p>Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the <code>gateway-sender-id</code> attribute of the <code>&lt;region-attributes&gt;</code> element.</p>	null
id	<p>Stores the region attribute settings in the cache with this identifier. Once stored, the attributes can be retrieved using the region attribute <code>refid</code>.</p> <p><b>API:</b> <code>setId</code></p>	not set

Attribute	Description	Default
	<b>Example:</b> <pre>&lt;region-attributes     id="persistent-replicated"&gt; &lt;/region-attributes&gt;</pre>	
ignore-jta	<p>Boolean that determines whether operations on this region participate in active JTA transactions or ignore them and operate outside of the transactions. This is primarily used in cache loaders, writers, and listeners that need to perform non-transactional operations on a region, such as caching a result set.</p> <p><b>API:</b> <code>setIgnoreJTA</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     ignore-jta="true"&gt; &lt;/region-attributes&gt;</pre>	false
index-update-type	<p>Specifies whether region indexes are maintained synchronously with region modifications, or asynchronously in a background thread. In the <code>cache.xml</code> file, this is set as a value, asynchronous or synchronous, assigned to the <code>index-update-type</code> region attribute. Set this through the API by passing a boolean to the <code>setIndexMaintenanceSynchronous</code> method.</p> <p><b>API:</b> <code>setIndexMaintenanceSynchronous</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     index-update-type="asynchronous"&gt; &lt;/region-attributes&gt;</pre>	synchronous updates
initial-capacity	<p>Together with the <code>load-factor</code> region attribute, sets the initial parameters on the underlying <code>java.util.ConcurrentHashMap</code> used for storing region entries.</p> <p><b>API:</b> <code>setInitialCapacity</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     initial-capacity="20"&gt; &lt;/region-attributes&gt;</pre>	16
is-lock-grantor	<p>Determines whether this member defines itself as the lock grantor for the region at region creation time. This only specifies whether the member becomes lock grantor at creation and does not reflect the current state of the member's lock grantor status. The member's lock grantor status may change if another member subsequently defines the region with <code>is-lock-grantor</code> set to true. This attribute is only relevant for regions with global scope, as only they allow locking. It affects implicit and explicit locking.</p> <p><b>API:</b> <code>setLockGrantor</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     is-lock-grantor="true"&gt; &lt;/region-attributes&gt;</pre>	false

Attribute	Description	Default
load-factor	<p>Together with the initial-capacity region attribute, sets the initial parameters on the underlying <code>java.util.ConcurrentHashMap</code> used for storing region entries. This must be a floating point number between 0 and 1, inclusive.</p> <p> <b>Note:</b> Before you set this attribute, read the discussion of initial capacity and load factor, then see the Java API documentation for <code>java.util.ConcurrentHashMap</code>.</p> <p><b>API:</b> <code>setLoadFactor</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   load-factor="0.85"&gt; &lt;/region-attributes&gt;</pre>	.75
mirror-type	Deprecated	
multicast-enabled	<p>Boolean that specifies whether distributed operations on a region should use multicasting. To enable this, multicast must be enabled for the distributed system with the <code>mcast-port gemfire.properties</code> setting.</p> <p><b>API:</b> <code>setMulticastEnabled</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   multicast-enabled="true"&gt; &lt;/region-attributes&gt;</pre>	false
persist-backup	Deprecated	
pool-name	<p>Identifies the region as a client region and specifies the server pool the region is to use. The named pool must be defined in the client cache before the region is created. If this is not set, the region does not connect to the servers as a client region.</p> <p><b>API:</b> <code>setPoolName</code></p> <p><b>Examples:</b></p> <p>This declaration creates the region as a client region with a server pool named DatabasePool. This pool-name specification is required, as there are multiple pools in the client cache:</p> <pre>&lt;client-cache&gt;   &lt;pool name="DatabasePool"     subscription-enabled="true"&gt;     ...   &lt;/pool&gt;   &lt;pool &gt;     name="OtherPool"     subscription-enabled="true"&gt;     ...   &lt;/pool&gt;   &lt;region ...     &lt;region-attributes       pool-name="DatabasePool"&gt;     &lt;/region-attributes&gt;     ...   &lt;/region&gt; &lt;/client-cache&gt;</pre>	not set

Attribute	Description	Default
	<p>This declaration creates the region as a client region assigned the single pool that is defined for the client cache. Here the pool-name specification is implied to be the only pool that exists in the cache:</p> <pre data-bbox="499 333 1217 635">&lt;client-cache&gt;   &lt;pool     name="publisher"     subscription-enabled="true"&gt;     ...   &lt;/pool&gt;   &lt;region     name="myRegion"     refid="CACHING_PROXY"&gt;   &lt;/region&gt; &lt;/client-cache&gt;</pre>	
disk-store-name	<p>Assigns the region to the disk store with this name from the disk stores defined for the cache. Persist region data to disk by defining the region as persistent using the Shortcut Attribute Options or data-policy settings. Overflow data to disk by implementing LRU eviction-attributes with an action of overflow to disk. Each disk store defines the file system directories to use, how data is written to disk, and other disk storage maintenance properties. In addition, the <code>disk-synchronous</code> region attribute specifies whether writes are done synchronously or asynchronously.</p> <p><b>API:</b> <code>setDiskStoreName</code></p> <p><b>Example:</b></p> <pre data-bbox="499 973 1217 1064">&lt;region-attributes   disk-store-name="myStoreA" &gt; &lt;/region-attributes&gt;</pre>	null
disk-synchronous	<p>For regions that write to disk, boolean that specifies whether disk writes are done synchronously for the region.</p> <p><b>API:</b> <code>setDiskSynchronous</code></p> <p><b>Example:</b></p> <pre data-bbox="499 1269 1217 1360">&lt;region-attributes   disk-store-name="myStoreA"   disk-synchronous="true"&gt; &lt;/region-attributes&gt;</pre>	true
publisher	Deprecated	
refid	<p>Retrieves region shortcuts and user-defined named region attributes for attributes initialization</p> <p><b>API:</b> <code>setRefId</code></p> <p><b>Example:</b></p> <pre data-bbox="499 1600 1217 1776">&lt;region-attributes   refid="persistent-replicated"&gt;   &lt;!-- Override any stored   attribute settings that you   need to ... --&gt; &lt;/region-attributes&gt;</pre>	not set
scope	Definition: Determines how updates to region entries are distributed to the other caches in the distributed system where the region and entry are defined. Scope also determines whether to allow remote invocation of some of the region's event handlers,	distributed-no-ack

Attribute	Description	Default								
	<p>and whether to use region entry versions to provide consistent updates across replicated regions.</p> <p> <b>Note:</b> You can configure the most common of these options with GemFire's region shortcuts in <code>RegionShortcut</code> and <code>ClientRegionShortcut</code>.</p> <p> <b>Note:</b> Server regions that are not partitioned must be replicated with <code>distributed-ack</code> or <code>global</code> scope. The region shortcuts that specify <code>REPLICATE</code> have <code>distributed-ack</code> scope.</p> <p>Set one of the following scope values:</p> <table> <tbody> <tr> <td>local</td> <td>No distribution. The region is visible only to threads running inside the member.</td> </tr> <tr> <td>distributed-no-ack</td> <td>Events are distributed to remote caches with no acknowledgement required.</td> </tr> <tr> <td>distributed-ack</td> <td>Events are distributed to remote caches with receipt acknowledgement required. Region entry versions are used to provide consistent updates across members of the distributed system (and across all distributed systems when region updates are distributed across a WAN).</td> </tr> <tr> <td>global</td> <td>Events are distributed to remote caches with global locking to ensure distributed cache consistency.</td> </tr> </tbody> </table> <p><b>API:</b> <code>setScope</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   scope="distributed-ack"&gt; &lt;/region-attributes&gt;</pre>	local	No distribution. The region is visible only to threads running inside the member.	distributed-no-ack	Events are distributed to remote caches with no acknowledgement required.	distributed-ack	Events are distributed to remote caches with receipt acknowledgement required. Region entry versions are used to provide consistent updates across members of the distributed system (and across all distributed systems when region updates are distributed across a WAN).	global	Events are distributed to remote caches with global locking to ensure distributed cache consistency.	
local	No distribution. The region is visible only to threads running inside the member.									
distributed-no-ack	Events are distributed to remote caches with no acknowledgement required.									
distributed-ack	Events are distributed to remote caches with receipt acknowledgement required. Region entry versions are used to provide consistent updates across members of the distributed system (and across all distributed systems when region updates are distributed across a WAN).									
global	Events are distributed to remote caches with global locking to ensure distributed cache consistency.									
statistics-enabled	<p>Boolean specifying whether to gather statistics on the region. Must be true to use expiration on the region. GemFire provides a standard set of statistics for cached regions and region entries, which give you information for fine-tuning your distributed system. Unlike other GemFire statistics, statistics for local and distributed regions are not archived and cannot be charted. They are kept in instances of <code>com.gemstone.gemfire.cache.CacheStatistics</code> and made available through the region and its entries through the <code>Region.getStatistics</code> and <code>Region.Entry.getStatistics</code> methods.</p> <p><b>API:</b> <code>setStatisticsEnabled</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   statistics-enabled="true"&gt; &lt;/region-attributes&gt;</pre>	false								
cloning-enabled	<p>Determines how <code>fromDelta</code> applies deltas to the local cache for delta propagation. When true, the updates are applied to a clone of the value and then the clone is saved to the cache. When false, the value is modified in place in the cache.</p> <p><b>API:</b> <code>setCloningEnabled</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   cloning-enabled="true"&gt; &lt;/region-attributes&gt;</pre>	false								

Attribute	Description	Default
concurrency-checks-enabled	<p>Determines whether members perform checks to provide consistent handling for concurrent or out-of-order updates to distributed regions. See <a href="#">Consistency for Region Updates</a> on page 239 and <a href="#">Overhead for Consistency Checks</a> on page 241.</p> <p> <b>Note:</b> Applications that use a <code>client-cache</code> may want to disable concurrency checking in order to see all events for a region. GemFire server members can continue using concurrency checks for the region, but they will pass all events to the client cache. This configuration ensures that the client sees all events, but it does not prevent the client cache from becoming out-of-sync with the server cache.</p> <p><b>API:</b> <code>setConcurrencyChecksEnabled</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   concurrency-checks-enabled="true" &gt; &lt;/region-attributes&gt;</pre>	true

### <key-constraint>

Defines the type of object to be allowed for the region entry keys. This must be a fully-qualified class name. The attribute ensures that the keys for the region entries are all of the same class. If key-constraint is not used, the region's keys can be of any class. This attribute, along with value-constraint, is useful for querying and indexing because it provides object type information to the query engine.

 **Note:** Set the constraint in every cache where you create or update the region entries. For client/server and multi-site installations, match constraints between client and server and between distributed systems. The constraint is only checked in the cache that does the entry put or create operation. To avoid deserializing the object, the constraint is not checked when the entry is distributed to other caches.

**Default:** not set

**API:** `com.gemstone.gemfire.cache.RegionFactory.setKeyConstraint`

**Example:**

```
<region-attributes>
  <key-constraint>
    java.lang.String
  </key-constraint>
</region-attributes>
```

### <value-constraint>

Defines the type of object to be allowed for the region entry values. This must be a fully-qualified class name. If value constraint isn't used, the region's value can be of any class. This attribute, along with key-constraint, is useful for querying and indexing because it provides object type information to the query engine.

 **Note:** Set the constraint in every cache where you create or update the region entries. For client/server and multi-site installations, match constraints between client and server and between distributed systems. The constraint is only checked in the cache that does the entry put or create operation. To avoid deserializing the object, the constraint is not checked when the entry is distributed to other caches.

**Default:** not set

**API:** `com.gemstone.gemfire.cache.RegionFactory.setValueConstraint`

**Example:**

```
<region-attributes>
  <value-constraint>
    cacheRunner.Portfolio
  </value-constraint>
</region-attributes>
```

**<region-time-to-live>**

Expiration setting that specifies how long the region can remain in the cache without anyone accessing or updating it.

**Default:** not set - no expiration of this type

**API:** `com.gemstone.gemfire.cache.RegionFactory.setRegionTimeToLive`

**Example:**

```
<region-attributes
  statistics-enabled="true">
  <region-time-to-live>
    <expiration-attributes
      timeout="3600"
      action="local-destroy"/>
  </region-time-to-live>
</region-attributes>
```

**<expiration-attributes>**

Within the `entry-time-to-live` or `entry-idle-time` element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally or across the distributed system. Within the `region-time-to-live` or `region-idle-time` element, this element specifies the expiration rules for the entire region.

**API:** See APIs for `entry-time-to-live`, `entry-idle-time`, `region-time-to-live`, `region-idle-time`

**Table 69: Attributes**

Attribute	Description	Default								
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	0								
action	<p>Action that should take place when a region or an entry expires.</p> <p>Select one of the following expiration actions:</p> <table> <tr> <td>local-destroy</td> <td>Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.</td> </tr> <tr> <td>destroy</td> <td>Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the distributed system.</td> </tr> <tr> <td>invalidate</td> <td>Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the distributed system.</td> </tr> <tr> <td>local-invalidate</td> <td>Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.</td> </tr> </table>	local-destroy	Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.	destroy	Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the distributed system.	invalidate	Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the distributed system.	local-invalidate	Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.	invalidate
local-destroy	Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.									
destroy	Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the distributed system.									
invalidate	Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the distributed system.									
local-invalidate	Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.									

**Example:**

```
<region-attributes
    statistics-enabled="true">
    <entry-time-to-live>
        <expiration-attributes
            timeout="60"
            action="local-destroy" />
    </entry-time-to-live>
</region-attributes>
```

<custom-expiry>

Specifies the custom class that implements com.gemstone.gemfire.cache.CustomExpiry. You define this class in order to override the region-wide settings for specific entries. See [Configure Data Expiration](#) on page 252 for an example.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name> and <parameter>](#) on page 911.

**API:** com.gemstone.gemfire.cache.RegionFactory.setCustomEntryIdleTimeout, setCustomEntryTimeToLive

**Example:**

```
<region-attributes>
    <expiration-attributes
        timeout="60"
        action="local-destroy">
        <custom-expiry>
            <class-name>
                com.megaconglomerate.mypackage.MyClass
            </class-name>
        </custom-expiry>
    </region-attributes>
```

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name> and <parameter>](#) on page 911.

**<region-idle-time>**

Expiration setting that specifies how long the region can remain in the cache without anyone accessing it.



**Note:** To ensure reliable read behavior across the partitioned region, use region-time-to-live for region expiration instead of this setting.

**Default:** not set - no expiration of this type

**API:** com.gemstone.gemfire.cache.RegionFactory.setRegionIdleTimeout

**Example:**

```
<region-attributes statistics-enabled="true">
    <region-idle-time>
        <expiration-attributes
            timeout="3600"
            action="local-destroy" />
    </region-idle-time>
</region-attributes>
```

**<expiration-attributes>**

Within the entry-time-to-live or entry-idle-time element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally

or across the distributed system. Within the `region-time-to-live` or `region-idle-time` element, this element specifies the expiration rules for the entire region.

**API:** See APIs for `entry-time-to-live`, `entry-idle-time`, `region-time-to-live`, `region-idle-time`

**Table 70: Attributes**

Attribute	Description	Default
<code>timeout</code>	Number of seconds before a region or an entry expires. If <code>timeout</code> is not specified, it defaults to zero (which means no expiration).	0
<code>action</code>	Action that should take place when a region or an entry expires. Select one of the following expiration actions:  local-destroy      Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries. destroy              Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the distributed system. invalidate              Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the distributed system. local-invalidate      Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.	invalidate

**Example:**

```
<region-attributes
  statistics-enabled="true">
  <entry-time-to-live>
    <expiration-attributes
      timeout="60"
      action="local-destroy" />
  </entry-time-to-live>
</region-attributes>

<custom-expiry>
```

Specifies the custom class that implements `com.gemstone.gemfire.cache.CustomExpiry`. You define this class in order to override the region-wide settings for specific entries. See [Configure Data Expiration](#) on page 252 for an example.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name> and <parameter>](#) on page 911.

**API:** `com.gemstone.gemfire.cache.RegionFactory.setCustomEntryIdleTimeout`, `setCustomEntryTimeToLive`

**Example:**

```
<region-attributes>
  <expiration-attributes
    timeout="60"
    action="local-destroy">
    <custom-expiry>
      <class-name>
        com.megaconglomerate.mypackage.MyClass
      </class-name>
    </custom-expiry>
  </expiration-attributes>
</region-attributes>
```

```
</custom-expiry>
</region-attributes>
```

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

#### **<entry-time-to-live>**

Expiration setting that specifies how long the region's entries can remain in the cache without anyone accessing or updating them. See [<expiration-attributes>](#) for details.

**Default:** not set - no expiration of this type.

**API:** com.gemstone.gemfire.cache.RegionFactory.setEntryTimeToLive

#### **Example:**

```
<region-attributes
  statistics-enabled="true">
  <entry-time-to-live>
    <expiration-attributes
      timeout="60"
      action="local-destroy"/>
  </entry-time-to-live>
</region-attributes>
```

#### **<expiration-attributes>**

Within the entry-time-to-live or entry-idle-time element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally or across the distributed system. Within the region-time-to-live or region-idle-time element, this element specifies the expiration rules for the entire region.

**API:** See APIs for entry-time-to-live, entry-idle-time, region-time-to-live, region-idle-time

**Table 71: Attributes**

Attribute	Description	Default								
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	0								
action	<p>Action that should take place when a region or an entry expires.</p> <p>Select one of the following expiration actions:</p> <table> <tr> <td>local-destroy</td> <td>Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.</td> </tr> <tr> <td>destroy</td> <td>Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the distributed system.</td> </tr> <tr> <td>invalidate</td> <td>Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the distributed system.</td> </tr> <tr> <td>local-invalidate</td> <td>Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.</td> </tr> </table>	local-destroy	Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.	destroy	Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the distributed system.	invalidate	Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the distributed system.	local-invalidate	Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.	invalidate
local-destroy	Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.									
destroy	Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the distributed system.									
invalidate	Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the distributed system.									
local-invalidate	Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.									

#### **Example:**

```
<region-attributes
  statistics-enabled="true">
```

```

<entry-time-to-live>
    <expiration-attributes
        timeout="60"
        action="local-destroy" />
</entry-time-to-live>
</region-attributes>
```

#### <custom-expiry>

Specifies the custom class that implements `com.gemstone.gemfire.cache.CustomExpiry`. You define this class in order to override the region-wide settings for specific entries. See [Configure Data Expiration](#) on page 252 for an example.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**API:** `com.gemstone.gemfire.cache.RegionFactory.setCustomEntryIdleTimeout`, `setCustomEntryTimeToLive`

#### Example:

```

<region-attributes>
    <expiration-attributes
        timeout="60"
        action="local-destroy">
        <custom-expiry>
            <class-name>
                com.megaconglomerate.mypackage.MyClass
            </class-name>
        </custom-expiry>
    </region-attributes>
```

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

#### <entry-idle-time>

Expiration setting that specifies how long the region's entries can remain in the cache without anyone accessing them. See [<expiration-attributes>](#) for details.



**Note:** To ensure reliable read behavior across the partitioned region, use `entry-time-to-live` for entry expiration instead of this setting.

**API:** `com.gemstone.gemfire.cache.RegionFactory.setEntryIdleTimeout`

#### Example:

```

<region-attributes
    statistics-enabled="true">
    <entry-idle-time>
        <expiration-attributes
            timeout="60"
            action="local-invalidate" />
        </expiration-attributes>
    </entry-idle-time>
</region-attributes>
```

#### <expiration-attributes>

Within the `entry-time-to-live` or `entry-idle-time` element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally or across the distributed system. Within the `region-time-to-live` or `region-idle-time` element, this element specifies the expiration rules for the entire region.

**API:** See APIs for `entry-time-to-live`, `entry-idle-time`, `region-time-to-live`, `region-idle-time`

**Table 72: Attributes**

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	0
action	<p>Action that should take place when a region or an entry expires.</p> <p>Select one of the following expiration actions:</p> <ul style="list-style-type: none"> <li>local-destroy      Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.</li> <li>destroy      Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the distributed system.</li> <li>invalidate      Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the distributed system.</li> <li>local-invalidate      Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.</li> </ul>	invalidate

**Example:**

```
<region-attributes
  statistics-enabled="true">
  <entry-time-to-live>
    <expiration-attributes
      timeout="60"
      action="local-destroy"/>
  </entry-time-to-live>
</region-attributes>
```

<custom-expiry>

Specifies the custom class that implements `com.gemstone.gemfire.cache.CustomExpiry`. You define this class in order to override the region-wide settings for specific entries. See [Configure Data Expiration](#) on page 252 for an example.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**API:** `com.gemstone.gemfire.cache.RegionFactory.setCustomEntryIdleTimeout`, `setCustomEntryTimeToLive`

**Example:**

```
<region-attributes>
  <expiration-attributes
    timeout="60"
    action="local-destroy">
    <custom-expiry>
      <class-name>
        com.megaconglomerate.mypackage.MyClass
      </class-name>
```

```
</custom-expiry>
</region-attributes>
```

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

### **<disk-write-attributes>**

DEPRECATED as of GemFire version 6.5. See [<disk-store>](#) on page 880

#### **<asynchronous-writes>**

DEPRECATED as of GemFire version 6.5. See [<disk-store>](#) on page 880

#### **<synchronous-writes>**

DEPRECATED as of GemFire version 6.5. See [<disk-store>](#) on page 880

#### **<disk-dirs>**

<disk-dirs> as a sub-element of <region-attributes> is deprecated as of GemFire version 6.5. See the disk-store-name attribute of the [<region-attributes>](#) element.

#### **<partition-attributes>**

Defines the region as partitioned and controls partitioning behavior. This is set during the region creation in the first data store for the partitioned region.



**Note:** With the exception of local-max-memory, all members defining a partitioned region must use the same partition attribute settings.

**API:** `com.gemstone.gemfire.cache.RegionFactory.setPartitionAttributes`

**Table 73: Attributes**

Attribute	Description	Default
colocated-with	The full name of a region to colocate with this region. The named region must exist before this region is created.	null
local-max-memory	Maximum megabytes of memory set aside for this region in the local member. This is all memory used for this partitioned region - for primary buckets and any redundant copies. This value must be smaller than the Java settings for the initial or maximum JVM heap. When the memory use goes above this value, GemFire issues a warning, but operation continues. Besides setting the maximum memory to use for the member, this setting also tells GemFire how to balance the load between members where the region is defined. For example, if one member sets this value to twice the value of another member's setting, GemFire works to keep the ratio between the first and the second at two-to-one, regardless of how little memory the region consumes. This is a local parameter that applies only to the local member. A value of 0 disables local data caching.	90% (of local heap)
recovery-delay	Applies when <code>redundant-copies</code> is greater than zero. The number of milliseconds to wait after a member crashes before reestablishing redundancy for the region. A setting of -1 disables automatic recovery of redundancy after member failure.	-1
redundant-copies	Number of extra copies that the partitioned region must maintain for each entry. Range: 0-3. If you specify 1, this partitioned region maintains the original and one backup, for a total of two copies. A value of 0 disables redundancy.	0
startup-recovery-delay	Applies when <code>redundant-copies</code> is greater than zero. The number of milliseconds a newly started member should wait before trying to satisfy redundancy of region data stored on other members. A setting of -1 disables automatic recovery of redundancy after new members join.	0
total-max-memory	Maximum combined megabytes of memory to be used by all processes hosting this region for all copies, primary and redundant.	<code>Integer.MAX_VALUE</code>

Attribute	Description	Default
total-num-buckets	Total number of buckets or data storage areas allotted for the entire partitioned region in the distributed cache. As data moves from one member to another, the entries in a bucket move as one unit. This value should be a prime number at least four times the number of data stores. More buckets increases overhead, however, especially when redundant-copies = 2 or 3 .	113

**Example:**

```
<region-attributes>
  <partition-attributes
    redundant-copies="1"
    total-num-buckets= "613"/>
</region-attributes>
```

**<partition-resolver>**

Describes a custom PartitionResolver for a region.

**API:**

com.gemstone.gemfire.cache.PartitionAttributesFactory.setPartitionResolver

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See <class-name> and <parameter> on page 911.

**Table 74: Attributes**

Attribute	Description	Default
name	The name of this custom PartitionResolver.	

**Example:**

```
<region name="trades">
  <region-attributes>
    <partition-attributes>
      <partition-resolver name="TradesPartitionResolver">
        <class-name>myPackage.TradesPartitionResolver
        </class-name>
      </partition-resolver>
    </partition-attributes>
  </region-attributes>
</region>
```

**<partition-listener>**

Defines a custom PartitionListener for a partitioned region.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See <class-name> and <parameter> on page 911.

**API:** com.gemstone.gemfire.cache.PartitionAttributesFactory.PartitionListener

**Example:**

```
<partition-attributes redundant-copies="1">
  <partition-listener>
    <class-name>com.myCompany.ColoatingPartitionListener</class-name>

    <parameter name="viewRegions">
      <string>/customer/ViewA,/customer/ViewB</string>
    </parameter>
```

```
</partition-listener>
</partition-attributes>
```

#### *<fixed-partition-attributes>*

Describes a partition in a Fixed Partitioned Region.

#### **API:**

`com.gemstone.gemfire.cache.PartitionAttributesFactory.addFixedPartitionAttributes`

**Table 75: Attributes**

Attribute	Description	Default
partition-name	The name of this fixed partition.	
is-primary	Set to true if this partition is the primary partition.	false
num-buckets	The number of buckets assigned to this partition.	

#### **Example:**

```
<cache>
  <region name="Trades">
    <region-attributes>
      <partition-attributes
        redundant-copies="1">
        <partition-resolver name="QuarterFixedPartitionResolver">
          <fixed-partition-attributes
            partition-name="Q1"
            is-primary="true"/>
          <fixed-partition-attributes
            partition-name="Q3"
            is-primary="false"
            num-buckets="6"/>
        </partition-attributes>
      </region-attributes>
    </region>
  </cache>
```

#### *<membership-attributes>*

Establishes reliability requirements and behavior for a region. Use this to configure the region to require one or more membership roles to be running in the system for reliable access to the region. You can set up your own roles, such as producer or backup, specifying each role as a string. Membership attributes have no effect unless one or more required roles are specified.

**API:** `com.gemstone.gemfire.cache.RegionFactory.setMembershipAttributes`

**Table 76: Attributes**

Attribute	Description		Default								
loss-action	Set one of the following values to specify how access to the region is affected when one or more required roles are lost.	<table border="1"> <tr> <td>full-access</td><td>Access to the region is unaffected when required roles are missing.</td></tr> <tr> <td>limited-access</td><td>Only local access to the region is allowed when required roles are missing.</td></tr> <tr> <td>no-access</td><td>The region is unavailable when required roles are missing.</td></tr> <tr> <td>reconnect</td><td>Loss of required roles causes the entire cache to be closed.</td></tr> </table>	full-access	Access to the region is unaffected when required roles are missing.	limited-access	Only local access to the region is allowed when required roles are missing.	no-access	The region is unavailable when required roles are missing.	reconnect	Loss of required roles causes the entire cache to be closed.	no_access
full-access	Access to the region is unaffected when required roles are missing.										
limited-access	Only local access to the region is allowed when required roles are missing.										
no-access	The region is unavailable when required roles are missing.										
reconnect	Loss of required roles causes the entire cache to be closed.										

Attribute	Description		Default				
resumption-action	<p>Specifies how the region is affected by resumption of reliability when one or more missing required roles return to the distributed membership. Set one of the following values:</p> <table border="1"> <tr> <td>none</td><td>No special action takes place when reliability resumes.</td></tr> <tr> <td>reinitialize</td><td>Resumption of reliability causes the region to be cleared of all data and replicated regions will do a new getInitialImage operation to repopulate the region.</td></tr> </table>		none	No special action takes place when reliability resumes.	reinitialize	Resumption of reliability causes the region to be cleared of all data and replicated regions will do a new getInitialImage operation to repopulate the region.	reinitialize
none	No special action takes place when reliability resumes.						
reinitialize	Resumption of reliability causes the region to be cleared of all data and replicated regions will do a new getInitialImage operation to repopulate the region.						

**Example:**

```
<!-- If there is no "producer" member
     running, do not allow access to the region -->
<region-attributes>
  <membership-attributes
    loss-action="no-access"
    resumption-action="none">
    <required-role
      name="producer">
    </required-role>
  </membership-attributes>
</region-attributes>
```

**<required-role>**

Specifies a role that is required for reliable access to the region.

**API:** com.gemstone.gemfire.cache.MembershipAttributes

**Table 77: Attributes**

Attribute	Description	Default
name	The name of the required role.	

**Example:**

```
<membership-attributes
  loss-action="no-access"
  resumption-action="none">
  <required-role name="producer" />
</membership-attributes>
```

**<subscription-attributes>**

Specifies subscriber requirements and behavior for the region. There is one subscription attribute, interest-policy, that defines which distributed entry events are delivered to the local region.



**Note:** The interest policy determines which events are delivered, but the data-policy determines how the events are applied to the cache.

**API:** com.gemstone.gemfire.cache.RegionFactory.setSubscriptionAttributes

**Table 78: Attributes**

Attribute	Description	Default
interest-policy	The two <code>interest-policy</code> options are:  cache-content      The default, registers interest in events that are already in the local region. For partitioned regions, the local member must hold the primary copy of the entry's data.  all      Registers interest in events for all entries that are anywhere in the distributed or partitioned region, regardless of whether they are already present in the local cache.	cache-content

**Example:**

```
<region-attributes>
  <subscription-attributes
    interest-policy="all" />
</region-attributes>
```

**<cache-loader>**

An event-handler plug-in that allows you to program for cache misses. At most, one cache loader can be defined in each member for the region. For distributed regions, a cache loader may be invoked remotely from other members that have the region defined. When an entry get results in a cache miss in a region with a cache loader defined, the loader's `load` method is called. This method is usually programmed to retrieve data from an outside data source, but it can do anything required by your application.

For partitioned regions, if you want to have a cache loader, install an instance of the cache loader in every data store. Partitioned regions support partitioned loading, where each cache loader loads only the data entries in the local member. If data redundancy is configured, data is loaded only if the local member holds the primary copy.

**API:** `com.gemstone.gemfire.cache.RegionFactory.setCacheLoader`

**Example:**

```
<region-attributes>
  <cache-loader>
    <class-name>quickstart.SimpleCacheLoader</class-name>
  </cache-loader>
</region-attributes>
```

**<cache-writer>**

An event-handler plug-in that allows you to receive before-event notification for changes to the region and its entries. It also has the ability to abort events. At most, one cache writer can be defined in each member for the region. A cache writer may be invoked remotely from other members that have the region defined.

**API:** `com.gemstone.gemfire.cache.RegionFactory.setCacheWriter`

**Example:**

```
<region-attributes>
  <cache-writer>
    <class-name>quickstart.SimpleCacheWriter</class-name>
  </cache-writer>
</region-attributes>
```

**<cache-listener>**

An event-handler plug-in that receives after-event notification of changes to the region and its entries. Any number of cache listeners can be defined for a region in any member. GemFire offers several listener types with callbacks to handle data and process events. Depending on the `data-policy` and the `interest-policy`

subscription attributes, a cache listener may receive only events that originate in the local cache, or it may receive those events along with events that originate remotely.

Specify the Java class for the cache listener and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**API:** com.gemstone.gemfire.cache.RegionFactory.addCacheListener

**Example:**

```
<region-attributes>
  <cache-listener>
    <class-name>
      quickstart.SimpleCacheListener
    </class-name>
  </cache-listener>
</region-attributes>
```

**<eviction-attributes>**

Specifies whether and how to control a region's size. Size is controlled by removing least recently used (LRU) entries to make space for new ones. This may be done through destroy or overflow actions. You can configure your region for lru-heap-percentage with an eviction action of local-destroy using GemFire's stored region attributes.

**Default:** Uses the lru-entry-count algorithm.

**API:** com.gemstone.gemfire.cache.RegionFactory.setEvictionAttributes

**Example:**

```
<region-attributes>
  <eviction-attributes>
    <lru-entry-count
      maximum="1000"
      action="overflow-to-disk" />
  </eviction-attributes>
</region-attributes>
```

**<lru-entry-count>**

Using the maximum attribute, specifies maximum region capacity based on entry count.

**Table 79: Attributes**

Attribute	Description	Default				
action	<p>Set one of the following eviction actions:</p> <table> <tr> <td>local-destroy</td> <td>Entry is destroyed locally. Not available for replicated regions.</td> </tr> <tr> <td>overflow-to-disk</td> <td>Entry is overflowed to disk and the value set to null in memory. For partitioned regions, this provides the most reliable read behavior across the region.</td> </tr> </table>	local-destroy	Entry is destroyed locally. Not available for replicated regions.	overflow-to-disk	Entry is overflowed to disk and the value set to null in memory. For partitioned regions, this provides the most reliable read behavior across the region.	local-destroy
local-destroy	Entry is destroyed locally. Not available for replicated regions.					
overflow-to-disk	Entry is overflowed to disk and the value set to null in memory. For partitioned regions, this provides the most reliable read behavior across the region.					
maximum	The maximum number of entries allowed in a region.					

**<lru-heap-percentage>**

Runs evictions when the GemFire resource manager says to. The manager orders evictions when the total cache size is over the heap percentage limit specified in the manager configuration. You can declare a Java class that implements the ObjectSizer interface to measure the size of objects in the Region.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**Table 80: Attributes**

Attribute	Description	Default				
action	<p>Set one of the following eviction actions:</p> <table> <tr> <td>local-destroy</td> <td>Entry is destroyed locally. Not available for replicated regions.</td> </tr> <tr> <td>overflow-to-disk</td> <td>Entry is overflowed to disk and the value set to null in memory. For partitioned regions, this provides the most reliable read behavior across the region.</td> </tr> </table>	local-destroy	Entry is destroyed locally. Not available for replicated regions.	overflow-to-disk	Entry is overflowed to disk and the value set to null in memory. For partitioned regions, this provides the most reliable read behavior across the region.	local-destroy
local-destroy	Entry is destroyed locally. Not available for replicated regions.					
overflow-to-disk	Entry is overflowed to disk and the value set to null in memory. For partitioned regions, this provides the most reliable read behavior across the region.					

**<lrn-memory-size>**

Using the maximum attribute, specifies maximum region capacity based on the amount of memory used, in megabytes. You can declare a Java class that implements the ObjectSizer interface to measure the size of objects in the Region.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name> and <parameter>](#) on page 911.

**Table 81: Attributes**

Attribute	Description	Default				
action	<p>Set one of the following eviction actions:</p> <table> <tr> <td>local-destroy</td> <td>Entry is destroyed locally. Not available for replicated regions.</td> </tr> <tr> <td>overflow-to-disk</td> <td>Entry is overflowed to disk and the value set to null in memory. For partitioned regions, this provides the most reliable read behavior across the region.</td> </tr> </table>	local-destroy	Entry is destroyed locally. Not available for replicated regions.	overflow-to-disk	Entry is overflowed to disk and the value set to null in memory. For partitioned regions, this provides the most reliable read behavior across the region.	local-destroy
local-destroy	Entry is destroyed locally. Not available for replicated regions.					
overflow-to-disk	Entry is overflowed to disk and the value set to null in memory. For partitioned regions, this provides the most reliable read behavior across the region.					
maximum	The maximum amount of memory used in the region, in megabytes.					

**<jndi-bindings>**

Specifies the binding for a data-source used in transaction management. See [Configuring Database Connections Using JNDI](#) on page 404.

**Example:**

```
<jndi-bindings>
  <jndi-binding
    type="XA PooledDataSource"
    jndi-name="newDB2trans"
    init-pool-size="20"
    max-pool-size="100"
    idle-timeout-seconds="20"
    blocking-timeout-seconds="5"
    login-timeout-seconds="10"
    xa-datasource-class="org.apache.derby.jdbc.EmbeddedXADataSource"
    user-name="mitul"

    password="encrypted(83f0069202c571faf1ae6c42b4ad46030e4e31c17409e19a)">
      <config-property>
        <config-property-name>description</config-property-name>
        <config-property-type>java.lang.String</config-property-type>
        <config-property-value>pooled_transact</config-property-value>
      </config-property>
      <config-property>
        <config-property-name>databaseName</config-property-name>
```

```

<config-property-type>java.lang.String</config-property-type>
<config-property-value>newDB</config-property-value>
</config-property>
<config-property>

<config-property-name>vendor_specific_property1</config-property-name>
    <config-property-type>type</config-property-type>
    <config-property-value>value</config-property-value>
</config-property>
    .
    .
    </jndi-binding>
</jndi-bindings>

```

### **<jndi-binding>**

For every datasource that is bound to the JNDI tree, there should be one `<jndi-binding>` element. This element describes the property and the configuration of the datasource. GemFire uses the attributes of the `<jndi-binding>` element for configuration. Use the `<config-property>` element to configure properties for the datasource.

VMware recommends that you set the username and password with the `user-name` and `password` attributes rather than using the `<config-property>` element.

**Table 82: Attributes**

Attribute	Description	Default
blocking-timeout-seconds	The number of seconds that a connection remains associated with a transaction. If this value is exceeded, the connection is disassociated from the transaction.	120
conn-pooled-datasource-class	Java class used for the <code>PooledDataSource</code> type.	
connection-url	URL for connecting to the datasource.	
idle-timeout-seconds	The maximum number of seconds that a connection can remain idle in a pool. When this threshold is reached, the connection is removed.	600
init-pool-size	The initial pool size of a <code>PooledConnection</code> (an <code>XAConnection</code> or a non- <code>XAConnection</code> ).	10
jdbc-driver-class	Java class used for the <code>SimpleDataSource</code> type.	
jndi-name	The <code>jndi-name</code> attribute is the key binding parameter. If the value of <code>jndi-name</code> is a <code>DataSource</code> , it is bound as <code>java:/myDatabase</code> , where <code>myDatabase</code> is the name you assign to your data source. If the data source cannot be bound to JNDI at runtime, GemFire logs a warning.	
login-timeout-seconds	<p>The maximum number of seconds for which a thread seeking a connection from a connection pool may be blocked. If the thread is unable to obtain connection in the stipulated time, a <code>PoolException</code> is thrown</p> <p>If a connection is available the thread returns immediately.</p> <p>If an existing connection is not available and the maximum number of connections in the pool has not been reached, a new connection is created and the thread returns immediately with the connection.</p> <p>If a connection is not available, the thread blocks for the specified time while waiting for an available connection.</p>	30
managed-conn-factory-class	If the Resource Adapter is of type <code>ManagedDataSource</code> , this class becomes the source of the <code>PooledConnection</code> . (This class interface complies with the J2CA Java 2 Connector Architecture.)	
max-pool-size	The maximum size of the <code>PooledConnection</code> .	30

Attribute	Description		Default
password	Password to access the datasource.		
transaction-type	When the <code>type</code> attribute is set to <code>ManagedDataSource</code> , specifies the type of transaction. Set one of the following <code>transaction-type</code> s:		none
	XATransaction	Select this option when you want to use <code>aManagedConnection</code> interface with a Java Transaction Manager to define transaction boundaries. This option allows a <code>ManagedDataSource</code> to participate in a transaction with a GemFire cache.	
	NoTransaction	No transactional behavior is used.	
	LocalTransaction	Select this option when using a <code>ManagedDataSource</code> that is not managed by the Java Transaction manager.	
type	Set one of the following types:		none
	XAPooledDataSource	Pooled SQL connections. For this type, you must also set the <code>xa-datasource-class</code> attribute.	
	ManagedDataSource	JNDI binding type for the J2EE Connector Architecture (J2CA). <code>ManagedConnectionFactory</code> . For information on the <code>ManagedConnection</code> interface, See: <a href="http://docs.oracle.com/javaee/6/api/javax/resource/spi/ManagedConnection.html">http://docs.oracle.com/javaee/6/api/javax/resource/spi/ManagedConnection.html</a> .	
	PooledDataSource	Pooled SQL connections. For this type, you must also set the <code>conn-pooled-datasource-class</code> attribute.	
	SimpleDataSource	Single SQL connection. No pooling of SQL connections is done. Connections are generated on the fly and cannot be reused. For this type, you must also set the <code>jdbc-driver-class</code> attribute.	
user-name	User name to access to the datasource.		
xa-datasource-class	Java class used for the <code>XAPooledDataSource</code> type.		

### <config-property>

A configuration property of the datasource. Use the sub-elements to identify the name, datatype, and value of the property.

#### Default:

#### API:

#### Example:

```
<config-property>
  <config-property-name>vendor_specific_property1</config-property-name>
    <config-property-type>>type</config-property-type>
    <config-property-value>value</config-property-value>
  </config-property>
```

<config-property-name>

The name of this datasource property.

<config-property-type>

The data type of this datasource property.

<config-property-value>

The value of this datasource property.

### <region>

Defines a region in the cache. See <[region-attributes](#)> for more details on configuring regions. You can specify zero or more sub-regions within a region.

#### **Default:**

**API:** com.gemstone.gemfire.cache.RegionFactory or  
com.gemstone.gemfire.cache.ClientRegionFactory

**Table 83: Attributes**

Attribute	Description	Default
name	Specify the name for the region. See <a href="#">Creating a Region Through the cache.xml File</a> on page 148 for details.	
refid	Used to apply predefined attributes to the region being defined. If the nested "region-attributes" element has its own "refid", then it will cause the "refid" on the region to be ignored. The "refid" region attribute can be set to the name of a RegionShortcut or a ClientRegionShortcut. For more information, see <a href="#">Region Shortcuts and Custom Named Region Attributes</a> on page 150 and <a href="#">Storing and Retrieving Region Shortcuts and Custom Named Region Attributes</a> on page 152.	

#### **Example:**

```
<!--Using region shortcut-->
<region
  name="PartitionedRegion"
  refid="PARTITION_REDUNDANT">
...
</region>

<!-- Retrieving and storing attributes -->
<region-attributes
  id="myPartition"
  refid="PARTITION_REDUNDANT">
<partition-attributes
  local-max-memory="512"/>
</region-attributes>

<!-- Attributes are retrieved and applied in the first region -->
<region name="PartitionedRegion1" refid="myPartition"/>
```

See <[region-attributes](#)> for a complete listing of region attributes.

### <index>

Describes an index to be created on a region. The index node, if any, should all come immediately after the "region-attributes" node. The "name" attribute is a required field which identifies the name of the index. See [Working with Indexes](#) on page 358 for more information on indexes. You can create either a functional index with the <functional> sub-element or a primary-key index with the <primary-key> element.

#### **Default:**

**API:** com.gemstone.gemfire.cache.QueryService.createIndex,  
createKeyIndex, createHashIndex

**Table 84: Attributes**

Attribute	Description	Default
name	Required. Name of the index.	
from-clause	Specifies the collection(s) of objects that the index ranges over. The from-clause must only contain one and only one region path.	
expression	Specifies the lookup value of the index.	
imports	String containing the imports used to create the index. String should be specified in the query language syntax with each import statement separated by a semicolon. The imports statement provides packages and classes used in variable typing in the indexed and FROM expressions.	
key-index	True or false. Whether the index should be a key index. If true, the region key specified in the indexed expression is used to evaluate queries	
type	Possible values are "hash" or "range".	range

**Example:**

```
<region name=exampleRegion>
<region-attributes . . . >
</region-attributes>
<index
  name="myIndex"
  from-clause="/exampleRegion"
  expression="status"/>
<index
  name="myKeyIndex"
  from-clause="/exampleRegion"
  expression="id" key-index="true"/>
<index
  name="myHashIndex"
  from-clause="/exampleRegion p"
  expression="p.mktValue" type="hash" />
...
</region>
```

**<functional>**

Defines a functional index. Deprecated after GemFire 6.6.1. Use the `type` attribute of `<index>` instead.

**Table 85: Attributes**

Attribute	Description	Default
expression	Specifies the collection(s) of objects that the index ranges over. The from-clause must only contain one and only one region path.	
from-clause	Specifies the lookup value of the index.	
imports	String containing the imports used to create the index. String should be specified in the query language syntax with each import statement separated by a semicolon. The imports statement provides packages and classes used in variable typing in the indexed and FROM expressions.	

**<primary-key>**

Defines a primary-key index. Deprecated after GemFire 6.6.1. Use the `key-index` attribute of `<index>` instead

**Table 86: Attributes**

Attribute	Description	Default
field	The field used to create the primary-key index.	

**<entry>**

An "entry" element describes an entry to be added to a region. Note that if an entry with the given key already exists in the region, it will be replaced.

**Default:**

**API:** com.gemstone.gemfire.cache.Region.create, put, get, putAll, getAll

**Example:**

```
<region ...>
  <region-attributes ...>
    ...
  </region-attributes>
  <entry>
    <key><string>MyKey</string></key>
    <value><string>MyValue</string></value>
  </entry>
</region>
```

**<key>**

Required. Describes the key in a region entry. A key can contain either a <string> or a <declarable> sub-element.

**<string>**

Specifies a String to be placed in a Region entry.

**Example:**

```
<region ...>
  <region-attributes ...>
    ...
  </region-attributes>
  <entry>
    <key><string>MyKey</string></key>
    <value><string>MyValue</string></value>
  </entry>
</region>
```

**<declarable>**

Specifies a Declarable object to be placed in a Region entry.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**API:** Declarable

**Example:**

```
<parameter name="cacheserver">
  <declarable>

    <class-name>com.gemstone.gemfire.addon.cache.CacheServerInitializer</class-name>

    <parameter name="system.property.prefix">
      <string>cacheserver</string>
```

```
</parameter>
</declarable>
</parameter>
```

**<value>**

Required. Describes the value of a region entry. A **<value>** can contain either a **<string>** or a **<declarable>** sub-element.

**<string>**

Specifies a String to be placed in a Region entry.

**Example:**

```
<region ...>
  <region-attributes ...>
    ...
  </region-attributes>
  <entry>
    <key><string>MyKey</string></key>
    <value><string>MyValue</string></value>
  </entry>
</region>
```

**<declarable>**

Specifies a Declarable object to be placed in a Region entry.

Specify the Java class and its initialization parameters with the **<class-name>** and **<parameter>** sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**API:** Declarable**Example:**

```
<parameter name="cacheserver">
  <declarable>

    <class-name>com.gemstone.gemfire.addon.cache.CacheServerInitializer</class-name>

    <parameter name="system.property.prefix">
      <string>cacheserver</string>
    </parameter>
  </declarable>
</parameter>
```

**<region>**

When nested within a **<region>** element, defines a sub-region.

See [<region>](#) on page 905

**<vm-root-region>**

DEPRECATED

**<function-service>**

Configures the behavior of the function execution service.

**Example:**

```
<cache>
  ...
</region>
```

```
<function-service>
  <function>
    <class-name>com.myCompany.tradeService.cache.func.TradeCalc</class-name>
  </function>
</function-service>
```

### **<function>**

Defines a function for registration in the function service

Specify the Java class for the function and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

#### **Default:**

**API:** com.gemstone.gemfire.cache.execute.FunctionService

#### **Example:**

```
<function>
  <class-name>
    com.myCompany.tradeService.cache.func.TradeCalc
  </class-name>
</function>
```

### **<resource-manager>**

A memory monitor that tracks cache size as a percentage of total tenured heap and controls size by restricting access to the cache and prompting eviction of old entries from the cache. Used in conjunction with settings for JVM memory and Java garbage collection.

**API:** com.gemstone.gemfire.cache.control.ResourceManager

**Table 87: Attributes**

Attribute	Description	Default
critical-heap-percentage	Percentage of heap at or above which the cache is considered in danger of becoming inoperable due to garbage collection pauses or out of memory exceptions.	0
eviction-heap-percentage	<p>Set the percentage of heap at or above which the eviction should begin on Regions configured for HeapLRU eviction.</p> <p>Changing this value may cause eviction to begin immediately.</p> <p>Only one change to this attribute or critical heap percentage will be allowed at any given time and its effect will be fully realized before the next change is allowed. This feature requires additional VM flags to perform properly. See <code>setCriticalHeapPercentage( )</code> for details.</p>	<ul style="list-style-type: none"> <li>• 0, If no region is configured with heap eviction</li> <li>• If <code>critical-heap-percentage</code> is set to a non-zero value, 5% less than that value.</li> <li>• 80%, if <code>critical-heap-percentage</code> is not configured.</li> </ul>

#### **Example:**

```
<cache>
...
<resource-manager
  critical-heap-percentage="99.9"
  eviction-heap-percentage="85" />
```

```
...
</cache>
```

### **<serialization-registration>**

Set of serializer or instantiator tags to register customer DataSerializer extensions or DataSerializable implementations respectively.

#### **Example:**

```
<serialization-registration>
  <instantiator id="42">
    <class-name>com.xxx.MyClass</class-name>
  </instantiator>
</serialization-registration>
```

### **<serializer>**

Allows you to configure the DataSerializer for this GemFire member. It registers a custom class which extends DataSerializer to support custom serialization of non-modifiable object types inside GemFire.

Specify the Java class for the DataSerializer and its initialization parameters with the `<class-name>` sub-element.

**API:** You can also register a DataSerializer by using the `com.gemstone.gemfire.DataSerializer.register` API. Use the `com.gemstone.gemfire.Instantiator` API to register a DataSerializable implementation.

### **<instantiator>**

An Instantiator registers a custom class which implements the DataSerializable interface to support custom object serialization inside GemFire.

Specify the Java class and its initialization parameters with the `<class-name>` sub-element.

#### **API: DataSerializable**

You can also directly specify `<instantiator>` as a sub-element of `<cache>`. Use the `com.gemstone.gemfire.Instantiator` API to register a DataSerializable implementation as the serialization framework for the cache. The following table lists the attribute that can be specified for an `<instantiator>`.

**Table 88: Attributes**

Attribute	Description	Default
<code>id</code>	Required. ID that the Instantiator should associate with the DataSerializable type.	

### **<backup>**

Defines additional files or directories that should be backed up when the system wide backup command is invoked. Disk stores with persistent data are automatically backed up and do not need to be listed with this element.

**Example:**

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

**<initializer>**

Used to specify a callback class (and optionally its parameters) that will be run after the cache is initialized. This element can be specified for both server and client caches.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**Default:**

**API:** Declarable

**Example:**

```
<initializer>
  <class-name>MyInitializer</class-name>
    <parameter name="members">
      <string>2</string>
    </parameter>
</initializer>
```

**<declarable>**

Specifies a Declarable object to be placed in a Region entry.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**API:** Declarable

**Example:**

```
<parameter name="cacheserver">
  <declarable>

    <class-name>com.gemstone.gemfire.addon.cache.CacheServerInitializer</class-name>

    <parameter name="system.property.prefix">
      <string>cacheserver</string>
    </parameter>
  </declarable>
</parameter>
```

**<class-name> and <parameter>**

Specify the name of a Java class with the `<class-name>` sub-element.

Specify initialization parameters for the class using the `<parameter>` sub-element. Use the `name` attribute to specify the name of the parameter and specify its value in the content of the [<string>](#) on page 912 sub-element or by specifying a Java class with the [<declarable>](#) on page 912 sub-element.

**Example:**

The following transaction writer configuration example specifies a Java class named `com.company.data.MyTransactionWriter`. The class is initialized with a parameter named `URL` whose value is `jdbc:cloudscape:rmi:MyData`.

```
<transaction-writer>
  <class-name>com.company.data.MyTransactionWriter</class-name>
```

```

<parameter name="URL">
    <string>jdbc:cloudscape:rmi:MyData</string>
</parameter>
</transaction-writer>
```

**<declarable>**

Specifies a Declarable object to be placed in a Region entry.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**API:** Declarable

**Example:**

```

<parameter name="cacheserver">
    <declarable>

        <class-name>com.gemstone.gemfire.addon.cache.CacheServerInitializer</class-name>

        <parameter name="system.property.prefix">
            <string>cacheserver</string>
        </parameter>
    </declarable>
</parameter>
```

**<string>**

Specifies a String to be placed in a Region entry.

**Example:**

```

<region ...>
    <region-attributes ...>
        ...
    </region-attributes>
    <entry>
        <key><string>MyKey</string></key>
        <value><string>MyValue</string></value>
    </entry>
</region>
```

**<client-cache> Element Hierarchy**

This section shows the hierarchy of <client-cache> element sub-elements that you use to configure GemFire caches and clients.

For details, click on one of the elements.

```

<client-cache> Element Reference on page 915
    <cache-transaction-manager> on page 915
        <transaction-listener>
        <transaction-writer>
    <dynamic-region-factory> on page 916
        <disk-dir> on page 917
    <pool> on page 917
        <locator>
        <server>
    <disk-store> on page 920
        <disk-dirs>
```

```
<disk-dir>
<pdx> on page 921
<pdx-serializer>
  <class-name>
  <parameter>
    <string.
      <declarable>
<region-attributes> on page 922
  <key-constraint>
  <value-constraint>
  <region-time-to-live>
    <expiration-attributes>
      <custom-expiry>
        <class-name>
        <parameter>
          <string>
          <declarable>
  <region-idle-time>
    <expiration-attributes>
      <custom-expiry>
        <class-name>
        <parameter>
          <string>
          <declarable>
  <entry-time-to-live>
    <expiration-attributes>
      <custom-expiry>
        <class-name>
        <parameter>
          <string.
            <declarable>
  <entry-idle-time>
    <expiration-attributes>
      <custom-expiry>
        <class-name>
        <parameter>
          <string>
          <declarable>
  <disk-write-attributes>
    <asynchronous-writes>
    <synchronous-writes>
  <disk-dirs>
    <disk-dir>
  <partition-attributes>
    <partition-resolver>
      <class-name>
      <parameter>
        <string>
        <declarable>
    <partition-listener>
      <class-name>
      <parameter>
        <string>
        <declarable>
    <fixed-partition=attributes>
  <membership-attributes>
    <required-role>
  <subscription-attributes>
  <cache-loader>
    <class-name>
    <parameter>
```

```
        <string>
        <declarable>
<cache-writer>
    <class-name>
        <parameter>
            <string>
            <declarable>
<cache-listener>
    <class-name>
        <parameter>
            <string>
            <declarable>
<eviction-attributes>
    <lru-entry-count>
    <lru-heap-percentage>
        <class-name>
            <parameter>
                <string>
                <declarable>
<lru-memory-size>
    <class-name>
        <parameter>
            <string>
            <declarable>
<jndi-bindings> on page 942
<jndi-binding>
    <config-property>
        <config-property-name>
        <config-property-type>
        <config-property-value>
<region> on page 944
    <region-attributes>
    <index>
        <functional>
        <primary-key>
    <entry>
        <key>
            <string>
            <declarable>
        <value>
            <string>
            <declarable>
    <region>
<function-service> on page 955
    <function>
        <class-name>
        <parameter>
            <string>
            <declarable>
<resource-manager> on page 956
<serialization-registration> on page 957
    <serializer>
        <class-name>
        <instantiator>
            <class-name>
<initializer> on page 957
    <class-name>
    <parameter>
        <string>
        <declarable>
```

```
</client-cache>
```

## **<client-cache> Element Reference**

This section documents all `cache.xml` elements that you use to configure GemFire clients. All elements are sub-elements of the `<client-cache>` element.

For GemFire server configuration, see [<cache> Element Reference](#) on page 865.

**API:** com.gemstone.gemfire.cache.client.ClientCacheFactory and PoolFactory interfaces.

**Table 89: Attributes of <client-cache>**

Attribute	Definition	Default
copy-on-read	Boolean indicating whether entry value retrieval methods return direct references to the entry value objects in the cache (false) or copies of the objects (true).	False

**Example:**

```
<client-cache>
  <pool
    name="client"
    subscription-enabled="true">
    <locator host="localhost" port="41111"/>
  </pool>
  <region-attributes
    id="clientAttributes"
    pool-name="client"
    refid="CACHING_PROXY"/>
  <region name="root">
    <region-attributes scope="local"/>
    <region name="cs_region">
      <region-attributes refid="clientAttributes"/>
    </region>
  </region>
</client-cache>
```

## **<cache-transaction-manager>**

Specifies a transaction listener.

**API:** CacheTransactionManager

**Example:**

```
<cache search-timeout="60">
  <cache-transaction-manager>
    <transaction-listener>
      <class-name>com.company.data.MyTransactionListener</class-name>
      <parameter name="URL">
        <string>jdbc:cloudscape:rmi:MyData</string>
      </parameter>
    </transaction-listener>
    <transaction-listener>... </transaction-listener>
    <transaction-writer>
      <class-name>com.company.data.MyTransactionWriter</class-name>
      <parameter name="URL">
        <string>jdbc:cloudscape:rmi:MyData</string>
      </parameter>
    </transaction-writer>
  </cache-transaction-manager>
</cache>
```

```

        <parameter
          />
        </transaction-writer>
      </cache-transaction-manager> . . .
    </cache>
  
```

### **<transaction-listener>**

When a transaction ends, its thread calls the TransactionListener to perform the appropriate follow-up for successful commits, failed commits, or voluntary rollbacks.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

### **<transaction-writer>**

When you commit a transaction, a TransactionWriter can perform additional tasks, including aborting the transaction.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

### **<dynamic-region-factory>**

The `<dynamic-region-factory>` element configures a dynamic region factory for this cache. You can use this element to dynamically create regions in your application code. Use the `createDynamicRegion()` method of the `com.gemstone.gemfire.cache.DynamicRegionFactory` class in your Java code to dynamically create regions.



**Note:** You can not use this element to dynamically create *partitioned* regions.

Pivotal Inc. recommends that you use functions to dynamically create regions. See [Creating Regions Dynamically](#) on page 208.

The optional `<disk-dir>` sub-element specifies the directory to store the persistent files that are used for dynamic region bookkeeping. It defaults to the current directory.

Set the `pool-name` attribute to set the name of the connection pool used by client applications in a client/server cache configuration. Do not specify the `pool-name` attribute in servers or peers.

**API:** `com.gemstone.gemfire.cache.DynamicRegionFactory`

**Table 90: Attributes**

Attribute	Description	Default
<code>disable-persist-backup</code>	When set to <code>false</code> , the factory is persisted on disk.	<code>false</code>
<code>disable-register-interest</code>	When set to <code>false</code> , client regions created by the factory register interest in all keys in a corresponding server cache region.	<code>false</code>
<code>pool-name</code>	The name of a connection pool used by the client factory to communicate with the server-side factory.	<code>none</code>

#### **Example:**

```

<dynamic-region-factory
  pool-name=myPool>
  <disk-dir>/home/gemfire/myDiskdir</disk-dir>
</dynamic-region-factory>
  
```

### **<disk-dir>**

Specifies a region or disk store's disk directory.

**Table 91: Attributes**

Attribute	Description	Default
dir-size	Maximum amount of space to use for the disk store, in megabytes.	214748364 (2 petabytes)

**Example:**

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

**<disk-dir>**

Specifies a region or disk store's disk directory.

**Table 92: Attributes**

Attribute	Description	Default
dir-size	Maximum amount of space to use for the disk store, in megabytes.	214748364 (2 petabytes)

**Example:**

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

**<pool>**

Use for client caches. Defines a client's server pool used to communicate with servers running in a different distributed system.

**API:** com.gemstone.gemfire.cache.client.PoolFactory

**Table 93: Attributes**

Attribute	Description	Default
free-connection-timeout	Amount of time a thread will wait to get a pool connection before timing out with an exception. This timeout keeps threads from waiting indefinitely when the pool's <b>max-connections</b> has been reached and all connections in the pool are in use by other threads.	10000
idle-timeout	Maximum time, in milliseconds, a pool connection can stay open without being used when there are more than <b>min-connections</b> in the pool. Pings over the connection do not count as connection use. If set to -1, there is no idle timeout.	5000
load-conditioning-interval	Amount of time, in milliseconds, a pool connection can remain open before being eligible for silent replacement to a less-loaded server.	300000 (5 minutes)
max-connections	Maximum number of pool connections the pool can create. If the maximum connections are in use, an operation requiring a client-to-server connection blocks until a connection becomes available or the <b>free-connection-timeout</b> is reached. If set to -1, there is no maximum. The setting must indicate a cap greater than <b>min-connections</b> .	-1
	 <b>Note:</b> If you need to use this to cap your pool connections, you should disable the pool attribute <b>pr-single-hop-enabled</b> . Leaving single hop enabled can increase thrashing and lower performance.	

Attribute	Description	Default
min-connections	Minimum number of pool connections to keep available at all times. Used to establish the initial connection pool. If set to 0 (zero), no connection is created until an operation requires it. This number is the starting point, with more connections added later as needed, up to the <code>max-connection</code> setting. The setting must be an integer greater than or equal to 0.	1
multiuser-authentication	Used for installations with security where you want to accommodate multiple users within a single client. If set to true, the pool provides authorization for multiple user instances in the same client application, and each user accesses the cache through its own <code>RegionService</code> instance. If false, the client either uses no authorization or just provides credentials for the single client process.	false
name	Name of this pool. Used in the client region pool-name to assign this pool to a region in the client cache.   <b>Note:</b> This is a required property with no default setting.	none
ping-interval	How often to communicate with the server to show the client is alive, set in milliseconds. Pings are only sent when the ping-interval elapses between normal client messages.   <b>Note:</b> Set this lower than the server's <code>maximum-time-between-pings</code> .	10000
pr-single-hop-enabled	Setting used to improve access to partitioned region data in the servers. Indicates whether to use metadata about the partitioned region data storage locations to decide where to send some data requests. This allows a client to send a data operation directly to the server hosting the key. Without this, the client contacts any available server and that server contacts the data store. This is used only for operations that can be carried out on a server-by-server basis, like put, get, and destroy.	true
read-timeout	Maximum time, in milliseconds, for the client to wait for a response from a server.	10000
retry-attempts	Number of times to retry a client request before giving up. If one server fails, the pool moves to the next, and so on until it is successful or it hits this limit. If the available servers are fewer than this setting, the pool will retry servers that have already failed until it reaches the limit. If this is set to -1, the pool tries every available server once.	-1
server-group	Logical named server group to use from the pool. A null value uses the global server group to which all servers belong.   <b>Note:</b> This is only used when the <code>locator</code> list is defined.	null
socket-buffer-size	Size for socket buffers from the client to the server. Default: 32768.	32768
statistic-interval	Interval, in milliseconds, at which to send client statistics to the server. If set to -1, statistics are not sent.	-1
subscription-ack-interval	Time, in milliseconds, between messages to the primary server to acknowledge event receipt.   <b>Note:</b> Used only when <code>subscription-redundancy</code> is not '0' (zero).	500
subscription-enabled	Boolean indicating whether the server should connect back to the client and automatically sends server-side cache update information. Any bind address information for the client is automatically passed to the server for use in the callbacks.	false
subscription-message-tracking-timeout	Time-to-live, in milliseconds, for entries in the client's message tracking list.	900000 (15 minutes)

Attribute	Description	Default
subscription-redundancy	Number of servers to use as backup to the primary for highly available subscription queue management. If set to 0, none are used. If set to -1, all available servers are used.	0
thread-local-connections	Boolean specifying whether connections are sticky. True causes the connection to stick to the thread for multiple requests. False causes each connection to be returned to the pool after a request finishes. A sticky connection is returned to the pool when the thread releases it through the <code>Pool</code> method <code>releaseThreadLocalConnection</code> , when the <code>idle-timeout</code> is reached, or when the pool is destroyed.	false

**Example:**

```
<pool
  name="publisher"
  subscription-enabled="true">
  <locator
    host="myLocatorAddress1"
    port="12345"/>
  <locator
    host="myLocatorAddress2"
    port="45678"/>
</pool>
```

**<locator>**

Addresses and ports of the locators to connect to. You can define multiple locators for the pool.



**Note:** Provide a locator list or `server` list, but not both.

**API:** com.gemstone.gemfire.distributed.LocatorLauncher

**Table 94: Attributes**

Attribute	Description	Default
host	Hostname of the locator	
port	Port number of the locator	

**Example:**

```
<pool ...>
<locator
  host="myLocatorHost"
  port="12345"/>
```

**<server>**

Addresses and ports of the servers to connect to.



**Note:** Provide a server list or `locator` list, but not both.

**Default:**

**API:**

**Table 95: Attributes**

Attribute	Description	Default
host	Hostname of the server	
port	Port number of the server	

**Example:**

```
<server
  </pool>
```

**<disk-store>**

Defines a pool of one or more disk stores, which can be used by regions, client subscription queues, and gateway sender queues for WAN distribution.

**Default:** The cache default disk store, named "DEFAULT", is used when disk is used but no disk store is named.

**API:** com.gemstone.gemfire.cache.DiskStore

**Table 96: Attributes**

Attribute	Description	Default
name	The name of the Disk Store.	
auto-compact	Set to true to automatically compact the disk files.	
compaction-threshold	The threshold at which an oplog will become compactable. Until it reaches this threshold the oplog will not be compacted.  The threshold is a percentage in the range 0 to 100.	
allow-force-compaction	Set to true to allow disk compaction to be forced on this disk store.	
max-oplog-size	The maximum size, in megabytes, of an oplog (operation log) file.	
time-interval	The number of milliseconds that can elapse before unwritten data is written to disk.	
write-buffer-size	The size of the write buffer that this disk store uses when writing data to disk. Larger values may increase performance but use more memory. The disk store allocates one direct memory buffer of this size.	
queue-size	Maximum number of operations that can be asynchronously queued to be written to disk.	

**Example:**

```
<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>
```

**<disk-dirs>**

DEPRECATED as of GemFire version 6.5. See the `disk-store-name` attribute of the [`<region-attributes>`](#) on page 882 element.

**<disk-dir>**

Specifies a region or disk store's disk directory.

**Table 97: Attributes**

Attribute	Description	Default
dir-size	Maximum amount of space to use for the disk store, in megabytes.	214748364 (2 petabytes)

**Example:**

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

**<pdx>**

Specifies the configuration for the Portable Data eXchange (PDX) method of serialization.

**API:** com.gemstone.gemfire.cache.CacheFactory.setPdxReadSerialized, setPdxDiskStore, setPdxPersistent, setPdxIgnoreUnreadFields and com.gemstone.gemfire.cache.ClientCacheFactory.setPdxReadSerialized, setPdxDiskStore, setPdxPersistent, setPdxIgnoreUnreadFields

**Table 98: Attributes**

Attribute	Description	Default
read-serialized	Set it to true if you want PDX deserialization to produce a PdxInstance instead of an instance of the domain class.	
ignore-unread-fields	Set it to true if you do not want unread PDX fields to be preserved during deserialization. You can use this option to save memory. Set to true only in members that are only reading data from the cache.	
persistent	Set to true if you are using persistent regions or WAN gateways. This causes the PDX type information to be written to disk.	
disk-store-name	If using persistence, this attribute allows you to configure the disk store that the PDX type data will be stored in. By default, the default disk store is used.	

**Example:**

```
<cache>
  <pdx persistent="true" disk-store-name="myDiskStore">
    <pdx-serializer>
      <class-name>
        com.gemstone.gemfire.pdx.ReflectionBasedAutoSerializer
      </class-name>
      <parameter name="classes">
        <string>com.company.domain.DomainObject</string>
      </parameter>
    </pdx-serializer>
  </pdx>
  ...
</cache>
```

**<pdx-serializer>**

Allows you to configure the PdxSerializer for this GemFire member.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**Default:**

**API:** PdxSerializer

**Example:**

```
<cache>
```

**<region-attributes>**

Specifies a region attributes template that can be named (by `id`) and referenced (by `ref-id`) later in the `cache.xml` and through the API.

**API:** com.gemstone.gemfire.cache.RegionFactory or  
com.gemstone.gemfire.cache.ClientRegionFactory

**Table 99: Attributes**

Attribute	Description	Default				
concurrency-level	<p>Gives an estimate of the maximum number of application threads that will concurrently access a region entry at one time. This attribute does not apply to partitioned regions. This attribute helps GemFire optimize the use of system resources and reduce thread contention. This sets an initial parameter on the underlying <code>java.util.ConcurrentHashMap</code> used for storing region entries.</p> <p> <b>Note:</b> Before you modify this, read the concurrency level description, then see the Java API documentation for <code>java.util.ConcurrentHashMap</code>.</p> <p><b>API:</b> <code>setConcurrencyLevel</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   concurrency-level="10"&gt; &lt;/region-attributes&gt;</pre>	16 (threads)				
data-policy	<p>Specifies how the local cache handles data for a region. This setting controls behavior such as local data storage and region initialization.</p> <p> <b>Note:</b> Configure the most common options using the region shortcuts, <code>RegionShortcut</code> and <code>ClientRegionShortcut</code>. The default <code>data-policy</code> of <code>normal</code> specifies local cache storage. The empty policy specifies no local storage. In the region shortcuts, empty corresponds to the settings with the string <code>PROXY</code>. You can use an empty region for event delivery to and from the local cache without the memory overhead of data storage.</p> <p>You can specify the following data policies:</p> <table> <tr> <td>empty</td> <td>No data storage in the local cache. The region always appears empty. Use this for event delivery to and from the local cache without the memory overhead of data storage - zero-footprint producers that only distribute data to others and zero-footprint consumers that only see events. To receive events with this, set the region's <code>subscription-attributes interest-policy</code> to <code>all</code>.</td> </tr> <tr> <td>normal</td> <td>Data used locally (accessed with <code>get</code>s, stored with <code>put</code>s, etc.) is stored in the local cache. This policy allows the contents in the cache to differ from other caches.</td> </tr> </table>	empty	No data storage in the local cache. The region always appears empty. Use this for event delivery to and from the local cache without the memory overhead of data storage - zero-footprint producers that only distribute data to others and zero-footprint consumers that only see events. To receive events with this, set the region's <code>subscription-attributes interest-policy</code> to <code>all</code> .	normal	Data used locally (accessed with <code>get</code> s, stored with <code>put</code> s, etc.) is stored in the local cache. This policy allows the contents in the cache to differ from other caches.	normal
empty	No data storage in the local cache. The region always appears empty. Use this for event delivery to and from the local cache without the memory overhead of data storage - zero-footprint producers that only distribute data to others and zero-footprint consumers that only see events. To receive events with this, set the region's <code>subscription-attributes interest-policy</code> to <code>all</code> .					
normal	Data used locally (accessed with <code>get</code> s, stored with <code>put</code> s, etc.) is stored in the local cache. This policy allows the contents in the cache to differ from other caches.					

Attribute	Description	Default
	<p>partition Data is partitioned across local and remote caches using the automatic data distribution behavior of partitioned regions. Additional configuration is done in the <b>partition-attributes</b>.</p> <p>replicate The region is initialized with the data from other caches. After initialization, all events for the distributed region are automatically copied into the local region, maintaining a replica of the entire distributed region in the local cache. Operations that would cause the contents to differ with other caches are not allowed. This is compatible with local <b>SCOPE</b>, behaving the same as for normal.</p> <p>persistent-partition Behaves the same as <b>partition</b> and also persists data to disk.</p> <p>persistent-replicate Behaves the same as <b>replicate</b> and also persists data to disk.</p> <p>preloaded Initializes like a replicated region, then, once initialized, behaves like a normal region.</p> <p><b>API:</b> <code>setDataPolicy</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   data-policy="replicate"&gt; &lt;/region-attributes&gt;</pre> <p>This is similar to using a region shortcut with <b>refid</b>, however when you use the <b>REPLICATE</b> region shortcut, it automatically sets the region's scope to <b>distributed-ack</b>.</p> <pre>&lt;region-attributes   refid="REPLICATE"&gt; &lt;/region-attributes&gt;</pre> <p>If you use <b>data-policy</b>, you must set the scope explicitly.</p>	
early-ack	Deprecated	
enable-async-conflation	<p>For TCP/IP distributions between peers, specifies whether to allow aggregation of asynchronous messages sent by the producer member for the region. This is a special-purpose boolean attribute that applies only when asynchronous queues are used for slow consumers. A false value disables conflation so that all asynchronous messages are sent individually. This special-purpose attribute gives you extra control over peer-to-peer communication between distributed regions using TCP/IP. This attribute does not apply to client/server communication, multi-site communication, or to communication using the UDP unicast or IP multicast protocols.</p> <p> <b>Note:</b> To use this attribute, the <b>multicast-enabled</b> region attribute <b>disable-tcp</b> in <code>gemfire.properties</code> must be false (the default for both). In addition, asynchronous queues must be enabled for slow consumers, specified with the <b>async*</b> <code>gemfire</code> properties.</p> <p><b>API:</b> <code>setEnableAsyncConflation</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   enable-async-conflation="false"&gt; &lt;/region-attributes&gt;</pre>	true
enable-gateway	<p>Determines whether the gateway is enabled for the region. When set to true, events in the region are sent to the defined gateway hubs.</p> <p>Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the <b>gateway-sender-id</b> attribute of the <code>&lt;region-attributes&gt;</code> element.</p>	false

Attribute	Description	Default
enable-subscription-conflation	<p>Boolean for server regions that specifies whether the server can conflate its messages to the client. A true value enables conflation.</p> <p> <b>Note:</b> The client can override this setting with the <code>conflate-events</code> property in its <code>gemfire.properties</code>.</p> <p><b>API:</b> <code>setEnableSubscriptionConflation</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   enable-subscription-conflation="true"&gt; &lt;/region-attributes&gt;</pre>	false
gateway-sender-ids	<p>Specifies one or more gateway sender IDs to use for distributing region events to remote GemFire sites. Specify multiple IDs as a comma-separated list.</p> <p><b>API:</b> <code>addGatewaySenderId</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   gateway-sender-ids="nwsender,swsender"&gt; &lt;/region-attributes&gt;</pre>	not set
async-event-queue-ids	<p>Specifies one or more asynchronous event queues to use for distributing region events an <code>AsyncEventListener</code> implementation (for example, for write-behind cache event handling). Specify multiple IDs as a comma-separated list.</p> <p><b>API:</b> <code>addAsyncEventQueueId</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   async-event-queue-ids="customerqueue,ordersqueue"&gt; &lt;/region-attributes&gt;</pre>	not set
hub-id	<p>If the <code>enable-gateway</code> attribute is set to true, a comma-separated list of gateway hub IDs that receive events from the region.</p> <p>Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the <code>gateway-sender-id</code> attribute of the <code>&lt;region-attributes&gt;</code> element.</p>	null
id	<p>Stores the region attribute settings in the cache with this identifier. Once stored, the attributes can be retrieved using the region attribute <code>refid</code>.</p> <p><b>API:</b> <code>setId</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   id="persistent-replicated"&gt; &lt;/region-attributes&gt;</pre>	not set
ignore-jta	<p>Boolean that determines whether operations on this region participate in active JTA transactions or ignore them and operate outside of the transactions. This is primarily used in cache loaders, writers, and listeners that need to perform non-transactional operations on a region, such as caching a result set.</p> <p><b>API:</b> <code>setIgnoreJTA</code></p>	false

Attribute	Description	Default
	<b>Example:</b> <pre>&lt;region-attributes     ignore-jta="true"&gt; &lt;/region-attributes&gt;</pre>	
index-update-type	<p>Specifies whether region indexes are maintained synchronously with region modifications, or asynchronously in a background thread. In the <code>cache.xml</code> file, this is set as a value, asynchronous or synchronous, assigned to the <code>index-update-type</code> region attribute. Set this through the API by passing a boolean to the <code>setIndexMaintenanceSynchronous</code> method.</p> <p><b>API:</b> <code>setIndexMaintenanceSynchronous</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     index-update-type="asynchronous"&gt; &lt;/region-attributes&gt;</pre>	synchronous updates
initial-capacity	<p>Together with the <code>load-factor</code> region attribute, sets the initial parameters on the underlying <code>java.util.ConcurrentHashMap</code> used for storing region entries.</p> <p><b>API:</b> <code>setInitialCapacity</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     initial-capacity="20"&gt; &lt;/region-attributes&gt;</pre>	16
is-lock-grantor	<p>Determines whether this member defines itself as the lock grantor for the region at region creation time. This only specifies whether the member becomes lock grantor at creation and does not reflect the current state of the member's lock grantor status. The member's lock grantor status may change if another member subsequently defines the region with <code>is-lock-grantor</code> set to true. This attribute is only relevant for regions with <code>global</code> scope, as only they allow locking. It affects implicit and explicit locking.</p> <p><b>API:</b> <code>setLockGrantor</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     is-lock-grantor="true"&gt; &lt;/region-attributes&gt;</pre>	false
load-factor	<p>Together with the initial-capacity region attribute, sets the initial parameters on the underlying <code>java.util.ConcurrentHashMap</code> used for storing region entries. This must be a floating point number between 0 and 1, inclusive.</p> <p> <b>Note:</b> Before you set this attribute, read the discussion of initial capacity and load factor, then see the Java API documentation for <code>java.util.ConcurrentHashMap</code>.</p> <p><b>API:</b> <code>setLoadFactor</code></p>	.75

Attribute	Description	Default
	<b>Example:</b> <pre>&lt;region-attributes     load-factor="0.85"&gt; &lt;/region-attributes&gt;</pre>	
mirror-type	Deprecated	
multicast-enabled	<p>Boolean that specifies whether distributed operations on a region should use multicasting. To enable this, multicast must be enabled for the distributed system with the <code>mcast-port</code> <code>gemfire.properties</code> setting.</p> <p><b>API:</b> <code>setMulticastEnabled</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     multicast-enabled="true"&gt; &lt;/region-attributes&gt;</pre>	false
persist-backup	Deprecated	
pool-name	<p>Identifies the region as a client region and specifies the server pool the region is to use. The named pool must be defined in the client cache before the region is created. If this is not set, the region does not connect to the servers as a client region.</p> <p><b>API:</b> <code>setPoolName</code></p> <p><b>Examples:</b></p> <p>This declaration creates the region as a client region with a server pool named <code>DatabasePool</code>. This pool-name specification is required, as there are multiple pools in the client cache:</p> <pre>&lt;client-cache&gt;   &lt;pool name="DatabasePool"     subscription-enabled="true"&gt;     ...   &lt;/pool&gt;   &lt;pool &gt;     name="OtherPool"     subscription-enabled="true"&gt;     ...   &lt;/pool&gt;   &lt;region ...     &lt;region-attributes       pool-name="DatabasePool"&gt;     &lt;/region-attributes&gt;     ...   &lt;/region&gt; &lt;/client-cache&gt;</pre> <p>This declaration creates the region as a client region assigned the single pool that is defined for the client cache. Here the pool-name specification is implied to be the only pool that exists in the cache:</p> <pre>&lt;client-cache&gt;   &lt;pool     name="publisher"     subscription-enabled="true"&gt;     ...   &lt;/pool&gt;   &lt;region     name="myRegion"     refid="CACHING_PROXY"&gt;</pre>	not set

Attribute	Description	Default
	<pre>&lt;/region&gt; &lt;/client-cache&gt;</pre>	
disk-store-name	<p>Assigns the region to the disk store with this name from the disk stores defined for the cache. Persist region data to disk by defining the region as persistent using the Shortcut Attribute Options or data-policy settings. Overflow data to disk by implementing LRU eviction-attributes with an action of overflow to disk. Each disk store defines the file system directories to use, how data is written to disk, and other disk storage maintenance properties. In addition, the <code>disk-synchronous</code> region attribute specifies whether writes are done synchronously or asynchronously.</p> <p><b>API:</b> <code>setDiskStoreName</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   disk-store-name="myStoreA" &gt; &lt;/region-attributes&gt;</pre>	null
disk-synchronous	<p>For regions that write to disk, boolean that specifies whether disk writes are done synchronously for the region.</p> <p><b>API:</b> <code>setDiskSynchronous</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   disk-store-name="myStoreA"   disk-synchronous="true"&gt; &lt;/region-attributes&gt;</pre>	true
publisher	Deprecated	
refid	<p>Retrieves region shortcuts and user-defined named region attributes for attributes initialization</p> <p><b>API:</b> <code>setRefId</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   refid="persistent-replicated"&gt;   &lt;!-- Override any stored   attribute settings that you   need to ... --&gt; &lt;/region-attributes&gt;</pre>	not set
scope	<p>Definition: Determines how updates to region entries are distributed to the other caches in the distributed system where the region and entry are defined. Scope also determines whether to allow remote invocation of some of the region's event handlers, and whether to use region entry versions to provide consistent updates across replicated regions.</p> <p> <b>Note:</b> You can configure the most common of these options with GemFire's region shortcuts in <code>RegionShortcut</code> and <code>ClientRegionShortcut</code>.</p> <p> <b>Note:</b> Server regions that are not partitioned must be replicated with <code>distributed-ack</code> or <code>global</code> scope. The region shortcuts that specify <code>REPLICATE</code> have <code>distributed-ack</code> scope.</p>	distributed-no-ack

Attribute	Description	Default								
	<p>Set one of the following scope values:</p> <table> <tr> <td>local</td><td>No distribution. The region is visible only to threads running inside the member.</td></tr> <tr> <td>distributed-no-ack</td><td>Events are distributed to remote caches with no acknowledgement required.</td></tr> <tr> <td>distributed-ack</td><td>Events are distributed to remote caches with receipt acknowledgement required. Region entry versions are used to provide consistent updates across members of the distributed system (and across all distributed systems when region updates are distributed across a WAN).</td></tr> <tr> <td>global</td><td>Events are distributed to remote caches with global locking to ensure distributed cache consistency.</td></tr> </table> <p><b>API:</b> <code>setScope</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   scope="distributed-ack"&gt; &lt;/region-attributes&gt;</pre>	local	No distribution. The region is visible only to threads running inside the member.	distributed-no-ack	Events are distributed to remote caches with no acknowledgement required.	distributed-ack	Events are distributed to remote caches with receipt acknowledgement required. Region entry versions are used to provide consistent updates across members of the distributed system (and across all distributed systems when region updates are distributed across a WAN).	global	Events are distributed to remote caches with global locking to ensure distributed cache consistency.	
local	No distribution. The region is visible only to threads running inside the member.									
distributed-no-ack	Events are distributed to remote caches with no acknowledgement required.									
distributed-ack	Events are distributed to remote caches with receipt acknowledgement required. Region entry versions are used to provide consistent updates across members of the distributed system (and across all distributed systems when region updates are distributed across a WAN).									
global	Events are distributed to remote caches with global locking to ensure distributed cache consistency.									
statistics-enabled	<p>Boolean specifying whether to gather statistics on the region. Must be true to use expiration on the region. GemFire provides a standard set of statistics for cached regions and region entries, which give you information for fine-tuning your distributed system. Unlike other GemFire statistics, statistics for local and distributed regions are not archived and cannot be charted. They are kept in instances of <code>com.gemstone.gemfire.cache.CacheStatistics</code> and made available through the region and its entries through the <code>Region.getStatistics</code> and <code>Region.Entry.getStatistics</code> methods.</p> <p><b>API:</b> <code>setStatisticsEnabled</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   statistics-enabled="true"&gt; &lt;/region-attributes&gt;</pre>	false								
cloning-enabled	<p>Determines how <code>fromDelta</code> applies deltas to the local cache for delta propagation. When true, the updates are applied to a clone of the value and then the clone is saved to the cache. When false, the value is modified in place in the cache.</p> <p><b>API:</b> <code>setCloningEnabled</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   cloning-enabled="true"&gt; &lt;/region-attributes&gt;</pre>	false								
concurrency-checks-enabled	<p>Determines whether members perform checks to provide consistent handling for concurrent or out-of-order updates to distributed regions. See <a href="#">Consistency for Region Updates</a> on page 239 and <a href="#">Overhead for Consistency Checks</a> on page 241.</p> <p> <b>Note:</b> Applications that use a <code>client-cache</code> may want to disable concurrency checking in order to see all events for a region. GemFire server members can continue using concurrency checks for the region, but they will pass all events to the client cache. This configuration ensures that the client sees all events, but it does not prevent the client cache from becoming out-of-sync with the server cache.</p>	true								

Attribute	Description	Default
	<p><b>API:</b> setConcurrencyChecksEnabled</p> <p><b>Example:</b></p> <pre>&lt;region-attributes   concurrency-checks-enabled="true"&gt; &lt;/region-attributes&gt;</pre>	

### <key-constraint>

Defines the type of object to be allowed for the region entry keys. This must be a fully-qualified class name. The attribute ensures that the keys for the region entries are all of the same class. If key-constraint is not used, the region's keys can be of any class. This attribute, along with value-constraint, is useful for querying and indexing because it provides object type information to the query engine.



**Note:** Set the constraint in every cache where you create or update the region entries. For client/server and multi-site installations, match constraints between client and server and between distributed systems. The constraint is only checked in the cache that does the entry put or create operation. To avoid deserializing the object, the constraint is not checked when the entry is distributed to other caches.

**Default:** not set

**API:** com.gemstone.gemfire.cache.RegionFactory.setKeyConstraint

**Example:**

```
<region-attributes>
  <key-constraint>
    java.lang.String
  </key-constraint>
</region-attributes>
```

### <value-constraint>

Defines the type of object to be allowed for the region entry values. This must be a fully-qualified class name. If value constraint isn't used, the region's value can be of any class. This attribute, along with key-constraint, is useful for querying and indexing because it provides object type information to the query engine.



**Note:** Set the constraint in every cache where you create or update the region entries. For client/server and multi-site installations, match constraints between client and server and between distributed systems. The constraint is only checked in the cache that does the entry put or create operation. To avoid deserializing the object, the constraint is not checked when the entry is distributed to other caches.

**Default:** not set

**API:** com.gemstone.gemfire.cache.RegionFactory.setValueConstraint

**Example:**

```
<region-attributes>
  <value-constraint>
    cacheRunner.Portfolio
  </value-constraint>
</region-attributes>
```

### <region-time-to-live>

Expiration setting that specifies how long the region can remain in the cache without anyone accessing or updating it.

**Default:** not set - no expiration of this type

**API:** com.gemstone.gemfire.cache.RegionFactory.setRegionTimeToLive

**Example:**

```
<region-attributes
    statistics-enabled="true">
    <region-time-to-live>
        <expiration-attributes
            timeout="3600"
            action="local-destroy"/>
    </region-time-to-live>
</region-attributes>
```

*<expiration-attributes>*

Within the entry-time-to-live or entry-idle-time element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally or across the distributed system. Within the region-time-to-live or region-idle-time element, this element specifies the expiration rules for the entire region.

**API:** See APIs for entry-time-to-live, entry-idle-time, region-time-to-live, region-idle-time

**Table 100: Attributes**

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	0
action	Action that should take place when a region or an entry expires.  Select one of the following expiration actions:  local-destroy      Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries. destroy            Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the distributed system. invalidate        Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the distributed system. local-invalidate   Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.	invalidate

**Example:**

```
<region-attributes
    statistics-enabled="true">
    <entry-time-to-live>
        <expiration-attributes
            timeout="60"
            action="local-destroy"/>
    </entry-time-to-live>
</region-attributes>
```

*<custom-expiry>*

Specifies the custom class that implements com.gemstone.gemfire.cache.CustomExpiry. You define this class in order to override the region-wide settings for specific entries. See [Configure Data Expiration](#) on page 252 for an example.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name> and <parameter>](#) on page 911.

**API:** com.gemstone.gemfire.cache.RegionFactory.setCustomEntryIdleTimeout, setCustomEntryTimeToLive

**Example:**

```
<region-attributes>
  <expiration-attributes
    timeout="60"
    action="local-destroy">
    <custom-expiry>
      <class-name>
        com.megaconglomerate.mypackage.MyClass
      </class-name>
    </custom-expiry>
  </region-attributes>
```

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name> and <parameter>](#) on page 911.

**<region-idle-time>**

Expiration setting that specifies how long the region can remain in the cache without anyone accessing it.



**Note:** To ensure reliable read behavior across the partitioned region, use region-time-to-live for region expiration instead of this setting.

**Default:** not set - no expiration of this type

**API:** com.gemstone.gemfire.cache.RegionFactory.setRegionIdleTimeout

**Example:**

```
<region-attributes statistics-enabled="true">
  <region-idle-time>
    <expiration-attributes
      timeout="3600"
      action="local-destroy"/>
  </region-idle-time>
</region-attributes>
```

**<expiration-attributes>**

Within the entry-time-to-live or entry-idle-time element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally or across the distributed system. Within the region-time-to-live or region-idle-time element, this element specifies the expiration rules for the entire region.

**API:** See APIs for entry-time-to-live, entry-idle-time, region-time-to-live, region-idle-time

**Table 101: Attributes**

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	0

Attribute	Description	Default
action	<p>Action that should take place when a region or an entry expires.</p> <p>Select one of the following expiration actions:</p> <ul style="list-style-type: none"> <li>local-destroy      Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.</li> <li>destroy      Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the distributed system.</li> <li>invalidate      Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the distributed system.</li> <li>local-invalidate      Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.</li> </ul>	invalidate

**Example:**

```
<region-attributes
    statistics-enabled="true">
    <entry-time-to-live>
        <expiration-attributes
            timeout="60"
            action="local-destroy" />
    </entry-time-to-live>
</region-attributes>
```

&lt;custom-expiry&gt;

Specifies the custom class that implements `com.gemstone.gemfire.cache.CustomExpiry`. You define this class in order to override the region-wide settings for specific entries. See [Configure Data Expiration](#) on page 252 for an example.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name> and <parameter>](#) on page 911.

**API:** `com.gemstone.gemfire.cache.RegionFactory.setCustomEntryIdleTimeout`, `setCustomEntryTimeToLive`

**Example:**

```
<region-attributes>
    <expiration-attributes
        timeout="60"
        action="local-destroy">
        <custom-expiry>
            <class-name>
                com.megaconglomerate.mypackage.MyClass
            </class-name>
        </custom-expiry>
    </region-attributes>
```

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name> and <parameter>](#) on page 911.

**<entry-time-to-live>**

Expiration setting that specifies how long the region's entries can remain in the cache without anyone accessing or updating them. See [<expiration-attributes>](#) on page 890 for details.

**Default:** not set - no expiration of this type.

**API:** com.gemstone.gemfire.cache.RegionFactory.setEntryTimeToLive

**Example:**

```
<region-attributes
  statistics-enabled="true">
  <entry-time-to-live>
    <expiration-attributes
      timeout="60"
      action="local-destroy"/>
  </entry-time-to-live>
</region-attributes>
```

**<expiration-attributes>**

Within the entry-time-to-live or entry-idle-time element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally or across the distributed system. Within the region-time-to-live or region-idle-time element, this element specifies the expiration rules for the entire region.

**API:** See APIs for entry-time-to-live, entry-idle-time, region-time-to-live, region-idle-time

**Table 102: Attributes**

Attribute	Description	Default								
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	0								
action	<p>Action that should take place when a region or an entry expires.</p> <p>Select one of the following expiration actions:</p> <table> <tr> <td>local-destroy</td> <td>Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.</td> </tr> <tr> <td>destroy</td> <td>Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the distributed system.</td> </tr> <tr> <td>invalidate</td> <td>Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the distributed system.</td> </tr> <tr> <td>local-invalidate</td> <td>Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.</td> </tr> </table>	local-destroy	Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.	destroy	Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the distributed system.	invalidate	Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the distributed system.	local-invalidate	Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.	invalidate
local-destroy	Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.									
destroy	Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the distributed system.									
invalidate	Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the distributed system.									
local-invalidate	Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.									

**Example:**

```
<region-attributes
  statistics-enabled="true">
  <entry-time-to-live>
    <expiration-attributes
      timeout="60"
      action="local-destroy"/>
  </entry-time-to-live>
</region-attributes>
```

**<custom-expiry>**

Specifies the custom class that implements com.gemstone.gemfire.cache.CustomExpiry. You define this class in order to override the region-wide settings for specific entries. See [Configure Data Expiration](#) on page 252 for an example.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**API:** com.gemstone.gemfire.cache.RegionFactory.setCustomEntryIdleTimeout, setCustomEntryTimeToLive

**Example:**

```
<region-attributes>
  <expiration-attributes
    timeout="60"
    action="local-destroy">
    <custom-expiry>
      <class-name>
        com.megaconglomerate.mypackage.MyClass
      </class-name>
    </custom-expiry>
  </region-attributes>
```

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**<entry-idle-time>**

Expiration setting that specifies how long the region's entries can remain in the cache without anyone accessing them. See [<expiration-attributes>](#) on page 890 for details.



**Note:** To ensure reliable read behavior across the partitioned region, use entry-time-to-live for entry expiration instead of this setting.

**API:** com.gemstone.gemfire.cache.RegionFactory.setEntryIdleTimeout

**Example:**

```
<region-attributes
  statistics-enabled="true">
  <entry-idle-time>
    <expiration-attributes
      timeout="60"
      action="local-invalidate" />
    </expiration-attributes>
  </entry-idle-time>
</region-attributes>
```

**<expiration-attributes>**

Within the entry-time-to-live or entry-idle-time element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally or across the distributed system. Within the region-time-to-live or region-idle-time element, this element specifies the expiration rules for the entire region.

**API:** See APIs for entry-time-to-live, entry-idle-time, region-time-to-live, region-idle-time

**Table 103: Attributes**

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	0

Attribute	Description	Default
action	<p>Action that should take place when a region or an entry expires.</p> <p>Select one of the following expiration actions:</p> <ul style="list-style-type: none"> <li>local-destroy      Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.</li> <li>destroy      Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the distributed system.</li> <li>invalidate      Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the distributed system.</li> <li>local-invalidate      Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.</li> </ul>	invalidate

**Example:**

```
<region-attributes
  statistics-enabled="true">
  <entry-time-to-live>
    <expiration-attributes
      timeout="60"
      action="local-destroy" />
  </entry-time-to-live>
</region-attributes>
```

<custom-expiry>

Specifies the custom class that implements com.gemstone.gemfire.cache.CustomExpiry. You define this class in order to override the region-wide settings for specific entries. See [Configure Data Expiration](#) on page 252 for an example.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name> and <parameter>](#) on page 911.

**API:** com.gemstone.gemfire.cache.RegionFactory.setCustomEntryIdleTimeout, setCustomEntryTimeToLive

**Example:**

```
<region-attributes>
  <expiration-attributes
    timeout="60"
    action="local-destroy">
    <custom-expiry>
      <class-name>
        com.megaconglomerate.mypackage.MyClass
      </class-name>
    </custom-expiry>
  </region-attributes>
```

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name> and <parameter>](#) on page 911.

**<disk-write-attributes>**

DEPRECATED as of GemFire version 6.5. See [<disk-store>](#) on page 880

**<asynchronous-writes>**

DEPRECATED as of GemFire version 6.5. See [<disk-store>](#) on page 880

**<synchronous-writes>**

DEPRECATED as of GemFire version 6.5. See [<disk-store>](#) on page 880

**<disk-dirs>**

DEPRECATED as of GemFire version 6.5. See the disk-store-name attribute of the [<region-attributes>](#) on page 882 element.

**<disk-dir>**

Specifies a region or disk store's disk directory.

**Table 104: Attributes**

Attribute	Description	Default
dir-size	Maximum amount of space to use for the disk store, in megabytes.	214748364 (2 petabytes)

**Example:**

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

**<partition-attributes>**

Defines the region as partitioned and controls partitioning behavior. This is set during the region creation in the first data store for the partitioned region.



**Note:** With the exception of local-max-memory, all members defining a partitioned region must use the same partition attribute settings.

**API:** com.gemstone.gemfire.cache.RegionFactory.setPartitionAttributes

**Table 105: Attributes**

Attribute	Description	Default
colocated-with	The full name of a region to colocate with this region. The named region must exist before this region is created.	null
local-max-memory	Maximum megabytes of memory set aside for this region in the local member. This is all memory used for this partitioned region - for primary buckets and any redundant copies. This value must be smaller than the Java settings for the initial or maximum JVM heap. When the memory use goes above this value, GemFire issues a warning, but operation continues. Besides setting the maximum memory to use for the member, this setting also tells GemFire how to balance the load between members where the region is defined. For example, if one member sets this value to twice the value of another member's setting, GemFire works to keep the ratio between the first and the second at two-to-one, regardless of how little memory the region consumes. This is a local parameter that applies only to the local member. A value of 0 disables local data caching.	90% (of local heap)
recovery-delay	Applies when redundant-copies is greater than zero. The number of milliseconds to wait after a member crashes before reestablishing redundancy for the region. A setting of -1 disables automatic recovery of redundancy after member failure.	-1

Attribute	Description	Default
redundant-copies	Number of extra copies that the partitioned region must maintain for each entry. Range: 0-3. If you specify 1, this partitioned region maintains the original and one backup, for a total of two copies. A value of 0 disables redundancy.	0
startup-recovery-delay	Applies when <code>redundant-copies</code> is greater than zero. The number of milliseconds a newly started member should wait before trying to satisfy redundancy of region data stored on other members. A setting of -1 disables automatic recovery of redundancy after new members join.	0
total-max-memory	Maximum combined megabytes of memory to be used by all processes hosting this region for all copies, primary and redundant.	<code>Integer.MAX_VALUE</code>
total-num-buckets	Total number of buckets or data storage areas allotted for the entire partitioned region in the distributed cache. As data moves from one member to another, the entries in a bucket move as one unit. This value should be a prime number at least four times the number of data stores. More buckets increases overhead, however, especially when <code>redundant-copies = 2 or 3</code> .	113

**Example:**

```
<region-attributes>
  <partition-attributes
    redundant-copies="1"
    total-num-buckets= "613" />
</region-attributes>
```

**<partition-resolver>**

Describes a custom PartitionResolver for a region.

**API:**

`com.gemstone.gemfire.cache.PartitionAttributesFactory.setPartitionResolver`

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**Table 106: Attributes**

Attribute	Description	Default
name	The name of this custom PartitionResolver.	

**Example:**

```
<region name="trades">
  <region-attributes>
    <partition-attributes>
      <partition-resolver name="TradesPartitionResolver">
        <class-name>myPackage.TradesPartitionResolver
        </class-name>
      </partition-resolver>
    </partition-attributes>
  </region-attributes>
</region>
```

**<partition-listener>**

Defines a custom PartitionListener for a partitioned region.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**API:** `com.gemstone.gemfire.cache.PartitionAttributesFactory.PartitionListener`

**Example:**

```
<partition-attributes redundant-copies="1">
    <partition-listener>
        <class-name>com.myCompany.ColoatingPartitionListener</class-name>

        <parameter name="viewRegions">
            <string>/customer/ViewA,/customer/ViewB</string>
        </parameter>
    </partition-listener>
</partition-attributes>
```

**<fixed-partition-attributes>**

Describes a partition in a Fixed Partitioned Region.

**API:**

`com.gemstone.gemfire.cache.PartitionAttributesFactory.addFixedPartitionAttributes`

**Table 107: Attributes**

Attribute	Description	Default
partition-name	The name of this fixed partition.	
is-primary	Set to true if this partition is the primary partition.	false
num-buckets	The number of buckets assigned to this partition.	

**Example:**

```
<cache>
    <region name="Trades">
        <region-attributes>
            <partition-attributes
                redundant-copies="1">
                <partition-resolver name="QuarterFixedPartitionResolver">
                    <fixed-partition-attributes
                        partition-name="Q1"
                        is-primary="true"/>
                    <fixed-partition-attributes
                        partition-name="Q3"
                        is-primary="false"
                        num-buckets="6"/>
                </partition-attributes>
            </region-attributes>
        </region>
    </cache>
```

**<membership-attributes>**

Establishes reliability requirements and behavior for a region. Use this to configure the region to require one or more membership roles to be running in the system for reliable access to the region. You can set up your own roles, such as producer or backup, specifying each role as a string. Membership attributes have no effect unless one or more required roles are specified.

**API:** `com.gemstone.gemfire.cache.RegionFactory.setMembershipAttributes`

**Table 108: Attributes**

Attribute	Description		Default
loss-action	Set one of the following values to specify how access to the region is affected when one or more required roles are lost.		no_access
	full-access	Access to the region is unaffected when required roles are missing.	
	limited-access	Only local access to the region is allowed when required roles are missing.	
	no-access	The region is unavailable when required roles are missing.	
	reconnect	Loss of required roles causes the entire cache to be closed.	
resumption-action	Specifies how the region is affected by resumption of reliability when one or more missing required roles return to the distributed membership. Set one of the following values:		reinitialize
	none	No special action takes place when reliability resumes.	
	reinitialize	Resumption of reliability causes the region to be cleared of all data and replicated regions will do a new getInitialImage operation to repopulate the region.	

**Example:**

```
<!-- If there is no "producer" member
     running, do not allow access to the region -->
<region-attributes>
  <membership-attributes>
    <required-role
      name="producer">
    </required-role>
  </membership-attributes>
</region-attributes>

<required-role>
```

Specifies a role that is required for reliable access to the region.

**API:** com.gemstone.gemfire.cache.MembershipAttributes

**Table 109: Attributes**

Attribute	Description	Default
name	The name of the required role.	

**Example:**

```
<membership-attributes
  loss-action="no-access"
  resumption-action="none">
  <required-role name="producer" />
</membership-attributes>
```

**<subscription-attributes>**

Specifies subscriber requirements and behavior for the region. There is one subscription attribute, `interest-policy`, that defines which distributed entry events are delivered to the local region.



**Note:** The interest policy determines which events are delivered, but the data-policy determines how the events are applied to the cache.

**API:** com.gemstone.gemfire.cache.RegionFactory.setSubscriptionAttributes

**Table 110: Attributes**

Attribute	Description	Default
interest-policy	<p>The two <code>interest-policy</code> options are:</p> <ul style="list-style-type: none"> <li>cache-content      The default, registers interest in events only for entries that are already in the local region. For partitioned regions, the local member must hold the primary copy of the entry's data.</li> <li>all                  Registers interest in events for all entries that are anywhere in the distributed or partitioned region, regardless of whether they are already present in the local cache.</li> </ul>	cache-content

**Example:**

```
<region-attributes>
  <subscription-attributes
    interest-policy="all" />
</region-attributes>
```

**<cache-listener>**

An event-handler plug-in that receives after-event notification of changes to the region and its entries. Any number of cache listeners can be defined for a region in any member. GemFire offers several listener types with callbacks to handle data and process events. Depending on the `data-policy` and the `interest-policy` subscription attributes, a cache listener may receive only events that originate in the local cache, or it may receive those events along with events that originate remotely.

Specify the Java class for the cache listener and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**API:** com.gemstone.gemfire.cache.RegionFactory.addCacheListener

**Example:**

```
<region-attributes>
  <cache-listener>
    <class-name>
      quickstart.SimpleCacheListener
    </class-name>
  </cache-listener>
</region-attributes>
```

**<eviction-attributes>**

Specifies whether and how to control a region's size. Size is controlled by removing least recently used (LRU) entries to make space for new ones. This may be done through destroy or overflow actions. You can configure your region for lru-heap-percentage with an eviction action of local-destroy using GemFire's stored region attributes.

**Default:** Uses the lru-entry-count algorithm.

**API:** com.gemstone.gemfire.cache.RegionFactory.setEvictionAttributes

**Example:**

```
<region-attributes>
  <eviction-attributes>
```

```

<lru-entry-count
    maximum="1000"
    action="overflow-to-disk" />
</eviction-attributes>
</region-attributes>

```

**<lru-entry-count>**

Using the maximum attribute, specifies maximum region capacity based on entry count.

**Table 111: Attributes**

Attribute	Description		Default
action	Set one of the following eviction actions:  local-destroy      Entry is destroyed locally. Not available for replicated regions. overflow-to-disk    Entry is overflowed to disk and the value set to null in memory. For partitioned regions, this provides the most reliable read behavior across the region.		local-destroy
maximum	The maximum number of entries allowed in a region.		

**<lru-heap-percentage>**

Runs evictions when the GemFire resource manager says to. The manager orders evictions when the total cache size is over the heap percentage limit specified in the manager configuration. You can declare a Java class that implements the ObjectSizer interface to measure the size of objects in the Region.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**Table 112: Attributes**

Attribute	Description		Default
action	Set one of the following eviction actions:  local-destroy      Entry is destroyed locally. Not available for replicated regions. overflow-to-disk    Entry is overflowed to disk and the value set to null in memory. For partitioned regions, this provides the most reliable read behavior across the region.		local-destroy

**<lru-memory-size>**

Using the maximum attribute, specifies maximum region capacity based on the amount of memory used, in megabytes. You can declare a Java class that implements the ObjectSizer interface to measure the size of objects in the Region.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**Table 113: Attributes**

Attribute	Description		Default
action	Set one of the following eviction actions:  local-destroy      Entry is destroyed locally. Not available for replicated regions.		local-destroy

Attribute	Description	Default
	overflow-to-disk Entry is overflowed to disk and the value set to null in memory. For partitioned regions, this provides the most reliable read behavior across the region.	
maximum	The maximum amount of memory used in the region, in megabytes.	

### <jndi-bindings>

Specifies the binding for a data-source used in transaction management. See [Configuring Database Connections Using JNDI](#) on page 404.

#### Example:

```
<jndi-bindings>
    <jndi-binding
        type="XA PooledDataSource"
        jndi-name="newDB2trans"
        init-pool-size="20"
        max-pool-size="100"
        idle-timeout-seconds="20"
        blocking-timeout-seconds="5"
        login-timeout-seconds="10"
        xa-datasource-class="org.apache.derby.jdbc.EmbeddedXADatasource"
        user-name="mitul"

    password="encrypted(83f0069202c571faf1ae6c42b4ad46030e4e31c17409e19a)">
        <config-property>
            <config-property-name>description</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
            <config-property-value>pooled_transact</config-property-value>
        </config-property>
        <config-property>
            <config-property-name>databaseName</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
            <config-property-value>newDB</config-property-value>
        </config-property>
        <config-property>

    <config-property-name>vendor_specific_property1</config-property-name>
        <config-property-type>type</config-property-type>
        <config-property-value>value</config-property-value>
    </config-property>
    .
    .
    </jndi-binding>
</jndi-bindings>
```

### <jndi-binding>

For every datasource that is bound to the JNDI tree, there should be one <jndi-binding> element. This element describes the property and the configuration of the datasource. GemFire uses the attributes of the <jndi-binding> element for configuration. Use the <config-property> element to configure properties for the datasource.

VMware recommends that you set the username and password with the user-name and password attributes rather than using the <config-property> element.

**Table 114: Attributes**

<b>Attribute</b>	<b>Description</b>	<b>Default</b>						
blocking-timeout-seconds	The number of seconds that a connection remains associated with a transaction. If this value is exceeded, the connection is disassociated from the transaction.	120						
conn-pooled-datasource-class	Java class used for the <code>PooledDataSource</code> type.							
connection-url	URL for connecting to the datasource.							
idle-timeout-seconds	The maximum number of seconds that a connection can remain idle in a pool. When this threshold is reached, the connection is removed.	600						
init-pool-size	The initial pool size of a <code>PooledConnection</code> (an <code>XAConnection</code> or a non- <code>XAConnection</code> ).	10						
jdbc-driver-class	Java class used for the <code>SimpleDataSource</code> type.							
jndi-name	The <code>jndi-name</code> attribute is the key binding parameter. If the value of <code>jndi-name</code> is a <code>DataSource</code> , it is bound as <code>java:/myDatabase</code> , where <code>myDatabase</code> is the name you assign to your data source. If the data source cannot be bound to JNDI at runtime, GemFire logs a warning.							
login-timeout-seconds	<p>The maximum number of seconds for which a thread seeking a connection from a connection pool may be blocked. If the thread is unable to obtain connection in the stipulated time, a <code>PoolException</code> is thrown</p> <p>If a connection is available the thread returns immediately.</p> <p>If an existing connection is not available and the maximum number of connections in the pool has not been reached, a new connection is created and the thread returns immediately with the connection.</p> <p>If a connection is not available, the thread blocks for the specified time while waiting for an available connection.</p>	30						
managed-conn-factory-class	If the Resource Adapter is of type <code>ManagedDataSource</code> , this class becomes the source of the <code>PooledConnection</code> . (This class interface complies with the J2CA Java 2 Connector Architecture.)							
max-pool-size	The maximum size of the <code>PooledConnection</code> .	30						
password	Password to access the datasource.							
transaction-type	<p>When the <code>type</code> attribute is set to <code>ManagedDataSource</code>, specifies the type of transaction. Set one of the following <code>transaction-types</code>:</p> <table border="1"> <tr> <td>XATransaction</td> <td>Select this option when you want to use a <code>ManagedConnection</code> interface with a Java Transaction Manager to define transaction boundaries. This option allows a <code>ManagedDataSource</code> to participate in a transaction with a GemFire cache.</td> </tr> <tr> <td>NoTransaction</td> <td>No transactional behavior is used.</td> </tr> <tr> <td>LocalTransaction</td> <td>Select this option when using a <code>ManagedDataSource</code> that is not managed by the Java Transaction manager.</td> </tr> </table>	XATransaction	Select this option when you want to use a <code>ManagedConnection</code> interface with a Java Transaction Manager to define transaction boundaries. This option allows a <code>ManagedDataSource</code> to participate in a transaction with a GemFire cache.	NoTransaction	No transactional behavior is used.	LocalTransaction	Select this option when using a <code>ManagedDataSource</code> that is not managed by the Java Transaction manager.	none
XATransaction	Select this option when you want to use a <code>ManagedConnection</code> interface with a Java Transaction Manager to define transaction boundaries. This option allows a <code>ManagedDataSource</code> to participate in a transaction with a GemFire cache.							
NoTransaction	No transactional behavior is used.							
LocalTransaction	Select this option when using a <code>ManagedDataSource</code> that is not managed by the Java Transaction manager.							
type	<p>Set one of the following types:</p> <table border="1"> <tr> <td>XAPooledDataSource</td> <td>Pooled SQL connections. For this type, you must also set the <code>xa-datasource-class</code> attribute.</td> </tr> <tr> <td>ManagedDataSource</td> <td>JNDI binding type for the J2EE Connector Architecture (J2CA). <code>ManagedConnectionFactory</code>. For information on the <code>ManagedConnection</code></td> </tr> </table>	XAPooledDataSource	Pooled SQL connections. For this type, you must also set the <code>xa-datasource-class</code> attribute.	ManagedDataSource	JNDI binding type for the J2EE Connector Architecture (J2CA). <code>ManagedConnectionFactory</code> . For information on the <code>ManagedConnection</code>	none		
XAPooledDataSource	Pooled SQL connections. For this type, you must also set the <code>xa-datasource-class</code> attribute.							
ManagedDataSource	JNDI binding type for the J2EE Connector Architecture (J2CA). <code>ManagedConnectionFactory</code> . For information on the <code>ManagedConnection</code>							

Attribute	Description		Default
		interface, See: <a href="http://docs.oracle.com/javaee/6/api/javax/resource/spi/ManagedConnection.html">http://docs.oracle.com/javaee/6/api/javax/resource/spi/ManagedConnection.html</a> .	
PooledDataSource	Pooled SQL connections. For this type, you must also set the <code>conn-pooled-datasource-class</code> attribute.		
SimpleDataSource	Single SQL connection. No pooling of SQL connections is done. Connections are generated on the fly and cannot be reused. For this type, you must also set the <code>jdbc-driver-class</code> attribute.		
user-name	User name to access to the datasource.		
xa-datasource-class	Java class used for the XAPooledDataSource type.		

### <config-property>

A configuration property of the datasource. Use the sub-elements to identify the name, datatype, and value of the property.

#### Default:

#### API:

#### Example:

```
<config-property>
<config-property-name>vendor_specific_property1</config-property-name>
    <config-property-type>>type</config-property-type>
    <config-property-value>value</config-property-value>
</config-property>
```

### <region>

Defines a region in the cache. See <[region-attributes](#)> on page 882 for more details on configuring regions. You can specify zero or more sub-regions within a region.

#### Default:

**API:** com.gemstone.gemfire.cache.RegionFactory or  
com.gemstone.gemfire.cache.ClientRegionFactory

**Table 115: Attributes**

Attribute	Description	Default
name	Specify the name for the region. See <a href="#">Creating a Region Through the cache.xml File</a> on page 148 for details.	
refid	Used to apply predefined attributes to the region being defined. If the nested "region-attributes" element has its own "refid", then it will cause the "refid" on the region to be ignored. The "refid" region attribute can be set to the name of a RegionShortcut or a ClientRegionShortcut. For more information, see <a href="#">Region Shortcuts and Custom Named Region Attributes</a> on page 150 and <a href="#">Storing and Retrieving Region Shortcuts and Custom Named Region Attributes</a> on page 152.	

#### Example:

```
<!--Using region shortcut-->
<region
    name="PartitionedRegion"
    refid="PARTITION_REDUNDANT">
```

```

...
</region>

<!-- Retrieving and storing attributes -->
<region-attributes
  id="myPartition"
  refid="PARTITION_REDUNDANT">
<partition-attributes
  local-max-memory="512"/>
</region-attributes>

<!-- Attributes are retrieved and applied in the first region -->
<region name="PartitionedRegion1" refid="myPartition"/>

```

See [<region-attributes>](#) on page 882 for a complete listing of region attributes.

### **<region-attributes>**

Specifies a region attributes template that can be named (by `id`) and referenced (by `refid`) later in the `cache.xml` and through the API.

**API:** `com.gemstone.gemfire.cache.RegionFactory` or  
`com.gemstone.gemfire.cache.ClientRegionFactory`

**Table 116: Attributes**

Attribute	Description	Default		
concurrency-level	<p>Gives an estimate of the maximum number of application threads that will concurrently access a region entry at one time. This attribute does not apply to partitioned regions. This attribute helps GemFire optimize the use of system resources and reduce thread contention. This sets an initial parameter on the underlying <code>java.util.ConcurrentHashMap</code> used for storing region entries.</p> <p> <b>Note:</b> Before you modify this, read the concurrency level description, then see the Java API documentation for <code>java.util.ConcurrentHashMap</code>.</p> <p><b>API:</b> <code>setConcurrencyLevel</code></p> <p><b>Example:</b></p> <pre> &lt;region-attributes   concurrency-level="10"&gt; &lt;/region-attributes&gt; </pre>	16 (threads)		
data-policy	<p>Specifies how the local cache handles data for a region. This setting controls behavior such as local data storage and region initialization.</p> <p> <b>Note:</b> Configure the most common options using the region shortcuts, <code>RegionShortcut</code> and <code>ClientRegionShortcut</code>. The default <code>data-policy</code> of <code>normal</code> specifies local cache storage. The <code>empty</code> policy specifies no local storage. In the region shortcuts, <code>empty</code> corresponds to the settings with the string <code>PROXY</code>. You can use an empty region for event delivery to and from the local cache without the memory overhead of data storage.</p> <p>You can specify the following data policies:</p> <table> <tr> <td>empty</td> <td>No data storage in the local cache. The region always appears empty. Use this for event delivery to and from the local cache without the memory overhead of data storage - zero-footprint producers that only distribute data to others and zero-footprint consumers that only see events. To receive events with this, set the</td> </tr> </table>	empty	No data storage in the local cache. The region always appears empty. Use this for event delivery to and from the local cache without the memory overhead of data storage - zero-footprint producers that only distribute data to others and zero-footprint consumers that only see events. To receive events with this, set the	normal
empty	No data storage in the local cache. The region always appears empty. Use this for event delivery to and from the local cache without the memory overhead of data storage - zero-footprint producers that only distribute data to others and zero-footprint consumers that only see events. To receive events with this, set the			

Attribute	Description	Default
	<p>region's <b>subscription-attributes interest-policy</b> to all.</p> <p>normal Data used locally (accessed with <code>get</code>s, stored with <code>put</code>s, etc.) is stored in the local cache. This policy allows the contents in the cache to differ from other caches.</p> <p>partition Data is partitioned across local and remote caches using the automatic data distribution behavior of partitioned regions. Additional configuration is done in the <b>partition-attributes</b>.</p> <p>replicate The region is initialized with the data from other caches. After initialization, all events for the distributed region are automatically copied into the local region, maintaining a replica of the entire distributed region in the local cache. Operations that would cause the contents to differ with other caches are not allowed. This is compatible with local <code>SCOPE</code>, behaving the same as for normal.</p> <p>persistent-partition Behaves the same as <code>partition</code> and also persists data to disk.</p> <p>persistent-replicate Behaves the same as <code>replicate</code> and also persists data to disk.</p> <p>preloaded Initializes like a replicated region, then, once initialized, behaves like a normal region.</p> <p><b>API:</b> <code>setDataPolicy</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     data-policy="replicate"&gt; &lt;/region-attributes&gt;</pre> <p>This is similar to using a region shortcut with <code>refid</code>, however when you use the <code>REPLICATE</code> region shortcut, it automatically sets the region's scope to <code>distributed-ack</code>.</p> <pre>&lt;region-attributes     refid="REPLICATE"&gt; &lt;/region-attributes&gt;</pre> <p>If you use <code>data-policy</code>, you must set the scope explicitly.</p>	
early-ack	Deprecated	
enable-async-conflation	<p>For TCP/IP distributions between peers, specifies whether to allow aggregation of asynchronous messages sent by the producer member for the region. This is a special-purpose boolean attribute that applies only when asynchronous queues are used for slow consumers. A false value disables conflation so that all asynchronous messages are sent individually. This special-purpose attribute gives you extra control over peer-to-peer communication between distributed regions using TCP/IP. This attribute does not apply to client/server communication, multi-site communication, or to communication using the UDP unicast or IP multicast protocols.</p> <p> <b>Note:</b> To use this attribute, the <code>multicast-enabled</code> region attribute <code>disable-tcp</code> in <code>gemfire.properties</code> must be false (the default for both). In addition, asynchronous queues must be enabled for slow consumers, specified with the <code>async*</code> <code>gemfire</code> properties.</p> <p><b>API:</b> <code>setEnableAsyncConflation</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     enable-async-conflation="false"&gt; &lt;/region-attributes&gt;</pre>	true

Attribute	Description	Default
enable-gateway	<p>Determines whether the gateway is enabled for the region. When set to true, events in the region are sent to the defined gateway hubs.</p> <p>Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the <code>gateway-sender-id</code> attribute of the <code>&lt;region-attributes&gt;</code> element.</p>	false
enable-subscription-conflation	<p>Boolean for server regions that specifies whether the server can conflate its messages to the client. A true value enables conflation.</p> <p> <b>Note:</b> The client can override this setting with the <code>conflate-events</code> property in its <code>gemfire.properties</code>.</p> <p><b>API:</b> <code>setEnableSubscriptionConflation</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   enable-subscription-conflation="true"&gt; &lt;/region-attributes&gt;</pre>	false
gateway-sender-ids	<p>Specifies one or more gateway sender IDs to use for distributing region events to remote GemFire sites. Specify multiple IDs as a comma-separated list.</p> <p><b>API:</b> <code>addGatewaySenderId</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   gateway-sender-ids="nwsender,swsender"&gt; &lt;/region-attributes&gt;</pre>	not set
async-event-queue-ids	<p>Specifies one or more asynchronous event queues to use for distributing region events an <code>AsyncEventListener</code> implementation (for example, for write-behind cache event handling). Specify multiple IDs as a comma-separated list.</p> <p><b>API:</b> <code>addAsyncEventQueueId</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   async-event-queue-ids="customerqueue,ordersqueue"&gt; &lt;/region-attributes&gt;</pre>	not set
hub-id	<p>If the <code>enable-gateway</code> attribute is set to true, a comma-separated list of gateway hub IDs that receive events from the region.</p> <p>Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the <code>gateway-sender-id</code> attribute of the <code>&lt;region-attributes&gt;</code> element.</p>	null
id	<p>Stores the region attribute settings in the cache with this identifier. Once stored, the attributes can be retrieved using the region attribute <code>refid</code>.</p> <p><b>API:</b> <code>setId</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   id="persistent-replicated"&gt; &lt;/region-attributes&gt;</pre>	not set

Attribute	Description	Default
ignore-jta	<p>Boolean that determines whether operations on this region participate in active JTA transactions or ignore them and operate outside of the transactions. This is primarily used in cache loaders, writers, and listeners that need to perform non-transactional operations on a region, such as caching a result set.</p> <p><b>API:</b> <code>setIgnoreJTA</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   ignore-jta="true"&gt; &lt;/region-attributes&gt;</pre>	false
index-update-type	<p>Specifies whether region indexes are maintained synchronously with region modifications, or asynchronously in a background thread. In the <code>cache.xml</code> file, this is set as a value, asynchronous or synchronous, assigned to the <code>index-update-type</code> region attribute. Set this through the API by passing a boolean to the <code>setIndexMaintenanceSynchronous</code> method.</p> <p><b>API:</b> <code>setIndexMaintenanceSynchronous</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   index-update-type="asynchronous"&gt; &lt;/region-attributes&gt;</pre>	synchronous updates
initial-capacity	<p>Together with the <code>load-factor</code> region attribute, sets the initial parameters on the underlying <code>java.util.ConcurrentHashMap</code> used for storing region entries.</p> <p><b>API:</b> <code>setInitialCapacity</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   initial-capacity="20"&gt; &lt;/region-attributes&gt;</pre>	16
is-lock-grantor	<p>Determines whether this member defines itself as the lock grantor for the region at region creation time. This only specifies whether the member becomes lock grantor at creation and does not reflect the current state of the member's lock grantor status. The member's lock grantor status may change if another member subsequently defines the region with <code>is-lock-grantor</code> set to true. This attribute is only relevant for regions with <code>global scope</code>, as only they allow locking. It affects implicit and explicit locking.</p> <p><b>API:</b> <code>setLockGrantor</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   is-lock-grantor="true"&gt; &lt;/region-attributes&gt;</pre>	false
load-factor	<p>Together with the initial-capacity region attribute, sets the initial parameters on the underlying <code>java.util.ConcurrentHashMap</code> used for storing region entries. This must be a floating point number between 0 and 1, inclusive.</p> <p> <b>Note:</b> Before you set this attribute, read the discussion of initial capacity and load factor, then see the Java API documentation for <code>java.util.ConcurrentHashMap</code>.</p>	.75

Attribute	Description	Default
	<p><b>API:</b> <code>setLoadFactor</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     load-factor="0.85"&gt; &lt;/region-attributes&gt;</pre>	
mirror-type	Deprecated	
multicast-enabled	<p>Boolean that specifies whether distributed operations on a region should use multicasting. To enable this, multicast must be enabled for the distributed system with the <code>mcast-port gemfire.properties</code> setting.</p> <p><b>API:</b> <code>setMulticastEnabled</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes     multicast-enabled="true"&gt; &lt;/region-attributes&gt;</pre>	false
persist-backup	Deprecated	
pool-name	<p>Identifies the region as a client region and specifies the server pool the region is to use. The named pool must be defined in the client cache before the region is created. If this is not set, the region does not connect to the servers as a client region.</p> <p><b>API:</b> <code>setPoolName</code></p> <p><b>Examples:</b></p> <p>This declaration creates the region as a client region with a server pool named <code>DatabasePool</code>. This pool-name specification is required, as there are multiple pools in the client cache:</p> <pre>&lt;client-cache&gt; &lt;pool name="DatabasePool"       subscription-enabled="true"&gt;   ... &lt;/pool&gt; &lt;pool &gt;   name="OtherPool"   subscription-enabled="true"&gt;   ... &lt;/pool&gt; &lt;region ...   &lt;region-attributes     pool-name="DatabasePool"&gt; &lt;/region-attributes&gt;   ...   ... &lt;/region&gt; &lt;/client-cache&gt;</pre> <p>This declaration creates the region as a client region assigned the single pool that is defined for the client cache. Here the pool-name specification is implied to be the only pool that exists in the cache:</p> <pre>&lt;client-cache&gt; &lt;pool   name="publisher"   subscription-enabled="true"&gt;   ... &lt;/pool&gt; &lt;region   name="myRegion"&gt; &lt;/region&gt; &lt;/client-cache&gt;</pre>	not set

Attribute	Description	Default
	<pre>refid="CACHING_PROXY"&gt; &lt;/region&gt; &lt;/client-cache&gt;</pre>	
disk-store-name	<p>Assigns the region to the disk store with this name from the disk stores defined for the cache. Persist region data to disk by defining the region as persistent using the Shortcut Attribute Options or data-policy settings. Overflow data to disk by implementing LRU eviction-attributes with an action of overflow to disk. Each disk store defines the file system directories to use, how data is written to disk, and other disk storage maintenance properties. In addition, the <b>disk-synchronous</b> region attribute specifies whether writes are done synchronously or asynchronously.</p> <p><b>API:</b> <code>setDiskStoreName</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   disk-store-name="myStoreA" &gt; &lt;/region-attributes&gt;</pre>	null
disk-synchronous	<p>For regions that write to disk, boolean that specifies whether disk writes are done synchronously for the region.</p> <p><b>API:</b> <code>setDiskSynchronous</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   disk-store-name="myStoreA"   disk-synchronous="true"&gt; &lt;/region-attributes&gt;</pre>	true
publisher	Deprecated	
refid	<p>Retrieves region shortcuts and user-defined named region attributes for attributes initialization</p> <p><b>API:</b> <code>setRefId</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   refid="persistent-replicated"&gt;   &lt;!-- Override any stored       attribute settings that you       need to ... --&gt; &lt;/region-attributes&gt;</pre>	not set
scope	<p>Definition: Determines how updates to region entries are distributed to the other caches in the distributed system where the region and entry are defined. Scope also determines whether to allow remote invocation of some of the region's event handlers, and whether to use region entry versions to provide consistent updates across replicated regions.</p> <p> <b>Note:</b> You can configure the most common of these options with GemFire's region shortcuts in <code>RegionShortcut</code> and <code>ClientRegionShortcut</code>.</p> <p> <b>Note:</b> Server regions that are not partitioned must be replicated with <code>distributed-ack</code> or <code>global</code> scope. The region shortcuts that specify <code>REPLICATE</code> have <code>distributed-ack</code> scope.</p>	distributed-no-ack

Attribute	Description	Default								
	<p>Set one of the following scope values:</p> <table> <tr> <td>local</td> <td>No distribution. The region is visible only to threads running inside the member.</td> </tr> <tr> <td>distributed-no-ack</td> <td>Events are distributed to remote caches with no acknowledgement required.</td> </tr> <tr> <td>distributed-ack</td> <td>Events are distributed to remote caches with receipt acknowledgement required. Region entry versions are used to provide consistent updates across members of the distributed system (and across all distributed systems when region updates are distributed across a WAN).</td> </tr> <tr> <td>global</td> <td>Events are distributed to remote caches with global locking to ensure distributed cache consistency.</td> </tr> </table> <p><b>API:</b> <code>setScope</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   scope="distributed-ack"&gt; &lt;/region-attributes&gt;</pre>	local	No distribution. The region is visible only to threads running inside the member.	distributed-no-ack	Events are distributed to remote caches with no acknowledgement required.	distributed-ack	Events are distributed to remote caches with receipt acknowledgement required. Region entry versions are used to provide consistent updates across members of the distributed system (and across all distributed systems when region updates are distributed across a WAN).	global	Events are distributed to remote caches with global locking to ensure distributed cache consistency.	
local	No distribution. The region is visible only to threads running inside the member.									
distributed-no-ack	Events are distributed to remote caches with no acknowledgement required.									
distributed-ack	Events are distributed to remote caches with receipt acknowledgement required. Region entry versions are used to provide consistent updates across members of the distributed system (and across all distributed systems when region updates are distributed across a WAN).									
global	Events are distributed to remote caches with global locking to ensure distributed cache consistency.									
statistics-enabled	<p>Boolean specifying whether to gather statistics on the region. Must be true to use expiration on the region. GemFire provides a standard set of statistics for cached regions and region entries, which give you information for fine-tuning your distributed system. Unlike other GemFire statistics, statistics for local and distributed regions are not archived and cannot be charted. They are kept in instances of <code>com.gemstone.gemfire.cache.CacheStatistics</code> and made available through the region and its entries through the <code>Region.getStatistics</code> and <code>Region.Entry.getStatistics</code> methods.</p> <p><b>API:</b> <code>setStatisticsEnabled</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   statistics-enabled="true"&gt; &lt;/region-attributes&gt;</pre>	false								
cloning-enabled	<p>Determines how <code>fromDelta</code> applies deltas to the local cache for delta propagation. When true, the updates are applied to a clone of the value and then the clone is saved to the cache. When false, the value is modified in place in the cache.</p> <p><b>API:</b> <code>setCloningEnabled</code></p> <p><b>Example:</b></p> <pre>&lt;region-attributes   cloning-enabled="true"&gt; &lt;/region-attributes&gt;</pre>	false								
concurrency-checks-enabled	<p>Determines whether members perform checks to provide consistent handling for concurrent or out-of-order updates to distributed regions. See <a href="#">Consistency for Region Updates</a> on page 239 and <a href="#">Overhead for Consistency Checks</a> on page 241.</p> <p> <b>Note:</b> Applications that use a <code>client-cache</code> may want to disable concurrency checking in order to see all events for a region. GemFire server members can continue using concurrency checks for the region, but they will pass all events to the client cache. This configuration ensures that the client sees all events, but it does not prevent the client cache from becoming out-of-sync with the server cache.</p>	true								

Attribute	Description	Default
	<p><b>API:</b> setConcurrencyChecksEnabled</p> <p><b>Example:</b></p> <pre>&lt;region-attributes   concurrency-checks-enabled="true"&gt; &lt;/region-attributes&gt;</pre>	

### <index>

Describes an index to be created on a region. The index node, if any, should all come immediately after the "region-attributes" node. The "name" attribute is a required field which identifies the name of the index. See [Working with Indexes](#) on page 358 for more information on indexes. You can create either a functional index with the <functional> sub-element or a primary-key index with the <primary-key> element.

#### Default:

**API:** com.gemstone.gemfire.cache.query.QueryService.createIndex, createKeyIndex, createHashIndex

**Table 117: Attributes**

Attribute	Description	Default
name	Required. Name of the index.	
from-clause	Specifies the collection(s) of objects that the index ranges over. The from-clause must only contain one and only one region path.	
expression	Specifies the lookup value of the index.	
imports	String containing the imports used to create the index. String should be specified in the query language syntax with each import statement separated by a semicolon. The imports statement provides packages and classes used in variable typing in the indexed and FROM expressions.	
key-index	True or false. Whether the index should be a key index. If true, the region key specified in the indexed expression is used to evaluate queries	
type	Possible values are "hash" or "range".	range

#### Example:

```
<region name=exampleRegion>
  <region-attributes . . . >
  </region-attributes>
  <index
    name="myIndex"
    from-clause="/exampleRegion"
    expression="status"/>
  <index
    name="myKeyIndex"
    from-clause="/exampleRegion"
    expression="id" key-index="true"/>
  <index
    name="myHashIndex"
    from-clause="/exampleRegion p"
    expression="p.mktValue" type="hash"/>
  ...
</region>
```

**<functional>**

Defines a functional index. Deprecated after GemFire 6.6.1. Use the `type` attribute of `<index>` instead.

**Table 118: Attributes**

Attribute	Description	Default
expression	Specifies the collection(s) of objects that the index ranges over. The from-clause must only contain one and only one region path.	
from-clause	Specifies the lookup value of the index.	
imports	String containing the imports used to create the index. String should be specified in the query language syntax with each import statement separated by a semicolon. The imports statement provides packages and classes used in variable typing in the indexed and FROM expressions.	

**<primary-key>**

Defines a primary-key index. Deprecated after GemFire 6.6.1. Use the `key-index` attribute of `<index>` instead

**Table 119: Attributes**

Attribute	Description	Default
field	The field used to create the primary-key index.	

**<entry>**

An "entry" element describes an entry to be added to a region. Note that if an entry with the given key already exists in the region, it will be replaced.

**Default:**

**API:** com.gemstone.gemfire.cache.Region.create, put, get, putAll, getAll

**Example:**

```
<region ...>
  <region-attributes ...>
    ...
  </region-attributes>
  <entry>
    <key><string>MyKey</string></key>
    <value><string>MyValue</string></value>
  </entry>
</region>
```

**<key>**

Required. Describes the key in a region entry. A key can contain either a `<string>` or a `<declarable>` sub-element.

**<string>**

Specifies a String to be placed in a Region entry.

**Example:**

```
<region ...>
  <region-attributes ...>
    ...
  </region-attributes>
  <entry>
```

```
<key><string>MyKey</string></key>
  <value><string>MyValue</string></value>
</entry>
</region>
```

## <declarable>

Specifies a Declarable object to be placed in a Region entry.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**API:** Declarable

**Example:**

```
<parameter name="cacheserver">
  <declarable>

    <class-name>com.gemstone.gemfire.addon.cache.CacheServerInitializer</class-name>

    <parameter name="system.property.prefix">
      <string>cacheserver</string>
    </parameter>
  </declarable>
</parameter>

<value>
```

Required. Describes the value of a region entry. A <value> can contain either a <string> or a <declarable> sub-element.

## <string>

Specifies a String to be placed in a Region entry.

**Example:**

```
<region ...>
  <region-attributes ...>
    ...
  </region-attributes>
  <entry>
    <key><string>MyKey</string></key>
    <value><string>MyValue</string></value>
  </entry>
</region>
```

## <declarable>

Specifies a Declarable object to be placed in a Region entry.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

**API:** Declarable

**Example:**

```
<parameter name="cacheserver">
  <declarable>

    <class-name>com.gemstone.gemfire.addon.cache.CacheServerInitializer</class-name>

    <parameter name="system.property.prefix">
      <string>cacheserver</string>
```

```
</parameter>
</declarable>
</parameter>
```

### **<region>**

Defines a region in the cache. See [<region-attributes>](#) on page 882 for more details on configuring regions. You can specify zero or more sub-regions within a region.

#### **Default:**

**API:** com.gemstone.gemfire.cache.RegionFactory or  
com.gemstone.gemfire.cache.ClientRegionFactory

**Table 120: Attributes**

Attribute	Description	Default
name	Specify the name for the region. See <a href="#">Creating a Region Through the cache.xml File</a> on page 148 for details.	
refid	Used to apply predefined attributes to the region being defined. If the nested "region-attributes" element has its own "refid", then it will cause the "refid" on the region to be ignored. The "refid" region attribute can be set to the name of a RegionShortcut or a ClientRegionShortcut. For more information, see <a href="#">Region Shortcuts and Custom Named Region Attributes</a> on page 150 and <a href="#">Storing and Retrieving Region Shortcuts and Custom Named Region Attributes</a> on page 152.	

#### **Example:**

```
<!--Using region shortcut-->
<region
  name="PartitionedRegion"
  refid="#PARTITION_REDUNDANT">
  ...
</region>

<!-- Retrieving and storing attributes -->
<region-attributes
  id="myPartition"
  refid="PARTITION_REDUNDANT">
  <partition-attributes
    local-max-memory="512"/>
</region-attributes>

<!-- Attributes are retrieved and applied in the first region -->
<region name="PartitionedRegion1" refid="myPartition"/>
```

See [<region-attributes>](#) on page 882 for a complete listing of region attributes.

### **<function-service>**

Configures the behavior of the function execution service.

#### **Example:**

```
<cache>
  ...
  </region>
<function-service>
  <function>
    <class-name>com.myCompany.tradeService.cache.func.TradeCalc</class-name>
```

```
</function>
</function-service>
```

### **<function>**

Defines a function for registration in the function service

Specify the Java class for the function and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

#### **Default:**

**API:** `com.gemstone.gemfire.cache.execute.FunctionService`

#### **Example:**

```
<function>
  <class-name>
    com.myCompany.tradeService.cache.func.TradeCalc
  </class-name>
</function>
```

### **<resource-manager>**

A memory monitor that tracks cache size as a percentage of total tenured heap and controls size by restricting access to the cache and prompting eviction of old entries from the cache. Used in conjunction with settings for JVM memory and Java garbage collection.

**API:** `com.gemstone.gemfire.cache.control.ResourceManager`

**Table 121: Attributes**

Attribute	Description	Default
<code>critical-heap-percentage</code>	Percentage of heap at or above which the cache is considered in danger of becoming inoperable due to garbage collection pauses or out of memory exceptions.	0
<code>eviction-heap-percentage</code>	<p>Set the percentage of heap at or above which the eviction should begin on Regions configured for HeapLRU eviction.</p> <p>Changing this value may cause eviction to begin immediately.</p> <p>Only one change to this attribute or critical heap percentage will be allowed at any given time and its effect will be fully realized before the next change is allowed. This feature requires additional VM flags to perform properly. See <code>setCriticalHeapPercentage( )</code> for details.</p>	<ul style="list-style-type: none"> <li>• 0, If no region is configured with heap eviction</li> <li>• If <code>critical-heap-percentage</code> is set to a non-zero value, 5% less than that value.</li> <li>• 80%, if <code>critical-heap-percentage</code> is not configured.</li> </ul>

#### **Example:**

```
<cache>
...
  <resource-manager
    critical-heap-percentage="99.9"
    eviction-heap-percentage="85" />
```

```
...
</cache>
```

### **<serialization-registration>**

Set of serializer or instantiator tags to register customer DataSerializer extensions or DataSerializable implementations respectively.

#### **Example:**

```
<serialization-registration>
    <instantiator id="42">
        <class-name>com.xxx.MyClass</class-name>
    </instantiator>
</serialization-registration>
```

### **<serializer>**

Allows you to configure the DataSerializer for this GemFire member. It registers a custom class which extends DataSerializer to support custom serialization of non-modifiable object types inside GemFire.

Specify the Java class for the DataSerializer and its initialization parameters with the `<class-name>` sub-element.

**API:** You can also register a DataSerializer by using the `com.gemstone.gemfire.DataSerializer.register` API. Use the `com.gemstone.gemfire.Instantiator` API to register a DataSerializable implementation.

### **<instantiator>**

An Instantiator registers a custom class which implements the DataSerializable interface to support custom object serialization inside GemFire.

Specify the Java class and its initialization parameters with the `<class-name>` sub-element.

#### **API:** DataSerializable

You can also directly specify `<instantiator>` as a sub-element of `<cache>`. Use the `com.gemstone.gemfire.Instantiator` API to register a DataSerializable implementation as the serialization framework for the cache. The following table lists the attribute that can be specified for an `<instantiator>`.

**Table 122: Attributes**

Attribute	Description	Default
id	Required. ID that the Instantiator should associate with the DataSerializable type.	

### **<initializer>**

Used to specify a callback class (and optionally its parameters) that will be run after the cache is initialized. This element can be specified for both server and client caches.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See [<class-name>](#) and [<parameter>](#) on page 911.

#### **Default:**

#### **API:** Declarable

**Example:**

```
<initializer>
  <class-name>MyInitializer</class-name>
    <parameter name="members">
      <string>2</string>
    </parameter>
</initializer>
```

# Chapter 66

---

## Region Shortcuts

Region shortcuts are groupings of pre-configured attributes that define the characteristics of a region. You can use a region shortcut as a starting point when configuring regions and you can add additional configurations to customize your application. To reference a region shortcut in a GemFire cache .xml file, use the `refid` attribute of the `<region>` element. For example:

```
<region name="myRegion" refid="PARTITION_REDUNDANT" />
```

You can override the default values and add additional configurations within a `<region-attributes>` element of the cache .xml file. For example, the following configuration overrides the local-max-memory setting and adds the recovery-delay attribute:

```
<region name="myRegion" refid="PARTITION_REDUNDANT">
  <region-attributes>
    <partition-attributes>
      local-max-memory="512"
      recovery-delay=-1/>
    </partition-attributes>
  </region-attributes>
</region>
```

You can also create your own, named region shortcuts for common custom configurations. See [Region Shortcuts and Custom Named Region Attributes](#) on page 150.

To configure a region using the gfsh command-line tool, specify the shortcut name with the `--type` argument. For example:

```
gfsh>create region --name=myRegion --type=PARTITION_REDUNDANT
```



**Note:** If you change the cache.xml file that defines a region, you must restart the member before the changes take effect.

For more information about configuring regions, see [Data Region Management](#) on page 145.

For more information about using the various types of GemFire regions and when to use them, see [Region Types](#) on page 206.

### Region Shortcuts Quick Reference

This section provides a quick reference for all region shortcuts.

[Table 123: Region Shortcuts Default Configurations](#) on page 960 summarizes the default configurations for each region shortcut. See the cross reference for additional information about each shortcut.

**Table 123: Region Shortcuts Default Configurations**

<b>Region Shortcut</b>	<b>Region Attributes</b>	<b>Other Attributes</b>
<a href="#">LOCAL</a> on page 963	scope: local data-policy: NORMAL	
<a href="#">LOCAL_HEAP_LRU</a> on page 964	scope: local data-policy: NORMAL	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: local-destroy
<a href="#">LOCAL_OVERFLOW</a> on page 965	scope: local data-policy: NORMAL	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk
<a href="#">LOCAL_PERSISTENT</a> on page 966	scope: local data-policy: PERSISTENT-REPLICATE	
<a href="#">LOCAL_PERSISTENT_OVERFLOW</a> on page 967	scope: local data-policy: PERSISTENT-REPLICATE	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk
<a href="#">PARTITION</a> on page 968	data-policy: PARTITION	
<a href="#">PARTITION_HEAP_LRU</a> on page 969	data-policy: PARTITION	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: local-destroy
<a href="#">PARTITION_OVERFLOW</a> on page 970	data-policy: PARTITION	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk
<a href="#">PARTITION_PERSISTENT</a> on page 971	data-policy: PERSISTENT-PARTITION	

Region Shortcut	Region Attributes	Other Attributes
<a href="#">PARTITION_PERSISTENT_OVERFLOW</a> on page 972	data-policy: PERSISTENT-PARTITION	Eviction Attributes  eviction-algorithm: lru-heap-percentage  eviction-action: overflow-to-disk
<a href="#">PARTITION_PROXY</a> on page 973	data-policy: PARTITION	Partition Attributes  local-max-memory: 0
<a href="#">PARTITION_PROXY_REDUNDANT</a> on page 974	data-policy: PARTITION	Partition Attributes  redundant-copies: 1  local-max-memory 0
<a href="#">PARTITION_REDUNDANT</a> on page 975	data-policy: PARTITION	Partition Attributes  redundant-copies: 1
<a href="#">PARTITION_REDUNDANT_HEAP_LRU</a> on page 976	data-policy: PARTITION	Eviction Attributes  eviction-algorithm: lru-heap-percentage  eviction-action: local-destroy  Partition Attributes  redundant-copies: 1
<a href="#">PARTITION_REDUNDANT_OVERFLOW</a> on page 977	data-policy: PARTITION	Eviction Attributes  eviction-algorithm: lru-heap-percentage  eviction-action: overflow-to-disk  Partition Attributes  redundant-copies: 1
<a href="#">PARTITION_REDUNDANT_PERSISTENT</a> on page 978	data-policy: PERSISTENT-PARTITION	Partition Attributes  redundant-copies: 1
<a href="#">PARTITION_REDUNDANT_PERSISTENT_OVERFLOW</a> on page 979	data-policy: PERSISTENT-PARTITION	Eviction Attributes  eviction-algorithm: lru-heap-percentage

Region Shortcut	Region Attributes	Other Attributes
		<p>eviction-action: <code>overflow-to-disk</code></p> <p>Partition Attributes</p> <p>redundant-copies: 1</p>
<a href="#">REPLICATE</a> on page 980	<p>data-policy: <code>REPLICATE</code></p> <p>scope: <code>distributed-ack</code></p>	
<a href="#">REPLICATE_HEAP_LRU</a> on page 981	<p>data-policy: <code>preloaded</code></p> <p>scope: <code>distributed-ack</code></p>	<p>Eviction Attributes</p> <p>eviction-algorithm: <code>lru-heap-percentage</code></p> <p>eviction-action: <code>local-destroy</code></p> <p> <b>Note:</b> Normally, you cannot have a <code>local-destroy</code> eviction policy on replicated regions. However, the <code>REPLICATE_HEAP_LRU</code> region is not the same as other replicated regions. This region is preloaded and remains in sync by registering interest in the keys of its peer's replicated regions.</p> <p>Subscription Attributes</p> <p>interest-policy: <code>all</code></p>
<a href="#">REPLICATE_OVERFLOW</a> on page 982	<p>data-policy: <code>replicate</code></p> <p>scope: <code>distributed-ack</code></p>	<p>Eviction Attributes</p> <p>eviction-algorithm: <code>lru-heap-percentage</code></p> <p>eviction-action: <code>overflow-to-disk</code></p>
<a href="#">REPLICATE_PERSISTENT</a> on page 983	<p>data-policy: <code>PERSISTENT-REPLICATE</code></p> <p>scope: <code>distributed-ack</code></p>	
<a href="#">REPLICATE_PERSISTENT_OVERFLOW</a> on page 984	<p>data-policy: <code>PERSISTENT-REPLICATE</code></p> <p>scope: <code>distributed-ack</code></p>	<p>Eviction Attributes</p> <p>eviction-algorithm: <code>lru-heap-percentage</code></p> <p>eviction-action: <code>overflow-to-disk</code></p>

Region Shortcut	Region Attributes	Other Attributes
<a href="#">REPLICATE_PROXY</a> on page 986	data-policy: EMPTY scope: distributed-ack	

## Region Shortcuts Reference

This topic describes the various region shortcuts you can use to configure GemFire regions.

### LOCAL

#### Description

A region configured with the LOCAL region shortcut is scoped only to its JVM and is not visible to other peer members. The region does not distribute data and operations to other caches.

For more information, see:

- [Local Regions](#) on page 208
- [RegionShortcuts for Peers and Servers](#) on page 151

#### Default Attributes

Region Attributes:

scope:	local
data-policy:	NORMAL

#### gfsh Command Example

```
gfsh>create region --name=myLOCALregion --type=LOCAL
```

```
gfsh>describe region --name=myLOCALregion
-----
Name          : myLOCALregion
Data Policy    : normal
Hosting Members : server1
                  server2
```

Non-Default Attributes Shared By Hosting Members

Type	Name	Value
Region	data-policy	NORMAL
	size	0

scope:	local
data-policy:	NORMAL

**cache.xml Example**

```
<region name="myLOCALregion">
  <region-attributes scope="local"/>
</region>
```

**LOCAL\_HEAP\_LRU****Description**

A region configured with the LOCAL\_HEAP\_LRU region shortcut is scoped to its JVM and is not visible to other peer members. The region does not distribute data and operations to other caches. The region destroys the least recently used entries when it detects that the JVM is running low on memory.

For more information, see:

- [Eviction](#) on page 248
- [Local Regions](#) on page 208

**Default Attributes**

## Region Attributes

scope:	local
data-policy:	NORMAL

## Eviction Attributes

eviction-algorithm:	lru-heap-percentage
eviction-action:	local-destroy

**gfsh Command Example**

```
gfsh>create region --name=myLHLregion --type=LOCAL_HEAP_LRU
```

```
gfsh>describe region --name=myLHLregion
-----
Name          : myLHLregion
Data Policy    : normal
Hosting Members : server1
                  server2
```

## Non-Default Attributes Shared By Hosting Members

Type	Name	Value
Region	data-policy	NORMAL
	size	0
Eviction	eviction-algorithm	lru-heap-percentage
	eviction-action	local-destroy

**cache.xml Example**

```
<region name="myLHLregion">
  <region-attributes
    scope="local">
    <eviction-attributes>
      <lru-heap-percentage>
```

```

        action="local-destroy">
<class-name>
  com.gemstone.gemfire.internal.size.SizeClassOnceObjectSizer
</class-name>
</lru-heap-percentage>
</eviction-attributes>
</region-attributes>
</region>

```

## LOCAL\_OVERFLOW

### Description

A region configured with the LOCAL\_OVERFLOW region shortcut is scoped only to its JVM and is not visible to other peer members. The region does not distribute data and operations to other caches. The region moves the values of entries to disk when it detects that the JVM is running low on memory.

For more information, see:

- [Persistence and Overflow](#) on page 245
- [Local Regions](#) on page 208
- [Eviction](#) on page 248

### Default Attributes

#### Region Attributes

scope:	local
data-policy:	NORMAL

#### Eviction Attributes

eviction-algorithm:	lru-heap-percentage
eviction-action:	overflow-to-disk

### gfsh Command Example

```
gfsh>create region --name=myLOregion --type=LOCAL_OVERFLOW
```

```
gfsh>describe region --name=myLOregion
-----
Name      : myLOregion
Data Policy   : normal
Hosting Members : server1
                  server2
```

#### Non-Default Attributes Shared By Hosting Members

Type	Name	Value
Region	data-policy	NORMAL
	size	0
Eviction	eviction-algorithm	lru-heap-percentage
	eviction-action	overflow-to-disk

**cache.xml Example**

```
<region name="myLOregion">
  <region-attributes
    scope="local">
    <eviction-attributes>
      <lru-heap-percentage
        action="overflow-to-disk"/>
    </eviction-attributes>
  </region-attributes>
</region>
```

**LOCAL\_PERSISTENT****Description**

A region configured with the LOCAL\_PERSISTENT region shortcut is scoped to its JVM and is not visible to other peer members. The region does not distribute data and operations to other caches. The region writes its state to disk and can recover that state when the member restarts.

For more information, see:

- [Persistence and Overflow](#) on page 245
- [Local Regions](#) on page 208

**Default Attributes**

Region Attributes

scope:	local
data-policy:	PERSISTENT-REPLICATE

**gfsh Command Example**

```
gfsh>create region --name=myLPregion --type=LOCAL_PERSISTENT
Member | Status
----- | -----
server2 | Region "/myLPregion" created on "server2"
server1 | Region "/myLPregion" created on "server1"

gfsh>describe region --name=myLPregion
.....
Name          : myLPregion
Data Policy   : persistent replicate
Hosting Members : server1
                  server2

Non-Default Attributes Shared By Hosting Members
```

Type	Name	Value
-----	-----	-----
Region	size	0

**cache.xml Example**

```
<region name="myLPregion">
  <region-attributes
    scope="local"
```

```
        data-policy="persistent-replicate"/>
</region>
```

## LOCAL\_PERSISTENT\_OVERFLOW

### Description

A region configured with the LOCAL\_PERSISTENT\_OVERFLOW region shortcut is scoped to its JVM and is not visible to other peer members. The region does not distribute data and operations to other caches. The region writes its state to disk and can recover that state when the member restarts. The region also removes the values of entries from memory when it detects that the JVM is running low on memory.

For more information, see:

- [Persistence and Overflow](#) on page 245
- [Local Regions](#) on page 208

### Default Attributes

Region Attributes

scope:	local
data-policy:	PERSISTENT-REPLICATE

Eviction Attributes

eviction-algorithm:	lru-heap-percentage
eviction-action:	overflow-to-disk

### gfsh Command Example

```
gfsh>create region --name=myLPOregion --type=LOCAL_PERSISTENT_OVERFLOW
Member | Status
----- | -----
server1 | Region "/myLPOregion" created on "server1"
server2 | Region "/myLPOregion" created on "server2"

gfsh>describe region --name=myLPOregion
Name : myLPOregion
Data Policy : persistent replicate
Hosting Members : server1
                  server2
```

Non-Default Attributes Shared By Hosting Members

Type	Name	Value
Region	size	0
Eviction	eviction-algorithm	lru-heap-percentage
	eviction-action	overflow-to-disk

### cache.xml Example

```
<region name="myLPOregion">
  <region-attributes
    scope="local"
```

```

        data-policy="persistent-replicate">
        <eviction-attributes>
            <lru-heap-percentage
                action="overflow-to-disk"/>
        </eviction-attributes>
    </region-attributes>
</region>
```

## PARTITION

### Description

A region configured with the PARTITION region shortcut has local scope that is partitioned across each peer member that created the region.

For more information, see:

- [Local Regions](#) on page 208
- [Partitioned Regions](#) on page 213
- [RegionShortcuts for Peers and Servers](#) on page 151

### Default Attributes

Region Attributes

data-policy:	PARTITION
--------------	-----------

### gfsh Command Example

```

gfsh>create region --name=myPregion --type=PARTITION
Member | Status
----- | -----
server1 | Region "/myPregion" created on "server1"
server2 | Region "/myPregion" created on "server2"

gfsh>describe region --name=myPregion
.....
Name : myPregion
Data Policy : partition
Hosting Members : server1
                  server2

Non-Default Attributes Shared By Hosting Members

Type | Name | Value
----- | ---- | -----
Region | size | 0
```

### cache.xml Example

```

<region name="myPregion">
    <region-attributes
        data-policy="partition"/>
</region>
```

## PARTITION\_HEAP\_LRU

### Description

A region configured with PARTITION\_HEAP\_LRU has local scope that is partitioned across each peer member that created the region. The region destroys entries when it detects that the JVM is running low on memory.

For more information, see:

- [Persistence and Overflow](#) on page 245
- [Local Regions](#) on page 208
- [Eviction](#) on page 248
- [Partitioned Regions](#) on page 213

### Default Attributes

Region Attributes

data-policy:	PARTITION
--------------	-----------

Eviction Attributes

eviction-algorithm:	lru-heap-percentage
eviction-action:	local-destroy

### gfsh Command Example

```
gfsh>create region --name=myPHLregion --type=PARTITION_HEAP_LRU
Member | Status
----- | -----
server1 | Region "/myPHLregion" created on "server1"
server2 | Region "/myPHLregion" created on "server2"

gfsh>describe region --name=myPHLregion
Name : myPHLregion
Data Policy : partition
Hosting Members : server1
                  server2

Non-Default Attributes Shared By Hosting Members

      Type   |       Name       |   Value
----- | ----- | -----
Region | size | 0
Eviction | eviction-algorithm | lru-heap-percentage
          | eviction-action | local-destroy
```

### cache.xml Example

```
<region name="myPHLregion">
  <region-attributes
    data-policy="partition">
    <eviction-attributes>
      <lru-heap-percentage
        action="local-destroy">
```

```
<class-name>com.gemstone.gemfire.internal.size.SizeClassOnceObjectSizer</class-name>
    </lru-heap-percentage>
    </eviction-attributes>
</region-attributes>
</region>
```

## PARTITION\_OVERFLOW

### Description

A region configured with the PARTITION\_OVERFLOW region shortcut has local state that is partitioned across each peer member that created the region. The region moves the values of entries to disk when it detects that the JVM is running low on memory.

For more information, see:

- [Persistence and Overflow](#) on page 245
- [Local Regions](#) on page 208
- [Partitioned Regions](#) on page 213
- [Eviction](#) on page 248

### Default Attributes

Region Attributes

data-policy:	PARTITION
--------------	-----------

Eviction Attributes

eviction-algorithm:	lru-heap-percentage
eviction-action:	overflow-to-disk

### gfsh Command Example

```
gfsh>create region --name=myPOregion --type=PARTITION_OVERFLOW
Member | Status
----- | -----
server1 | Region "/myPOregion" created on "server1"
server2 | Region "/myPOregion" created on "server2"

gfsh>describe region --name=myPOregion
Name : myPOregion
Data Policy : partition
Hosting Members : server1
                server2
```

Non-Default Attributes Shared By Hosting Members

Type	Name	Value
Region	size	0
Eviction	eviction-algorithm	lru-heap-percentage
	eviction-action	overflow-to-disk

### cache.xml Example

```
<region name="myPOregion">
    <region-attributes
        data-policy="partition">
        <eviction-attributes>
            <lru-heap-percentage
                action="overflow-to-disk" />
        </eviction-attributes>
    </region-attributes>
</region>
```

## PARTITION\_PERSISTENT

### Description

A region configured with the PARTITION\_PERSISTENT has local state that is partitioned across each peer member that created the region. The region writes its state to disk and can recover that state when the member restarts.

For more information, see:

- [RegionShortcuts for Peers and Servers](#) on page 151
- [Persistence and Overflow](#) on page 245
- [Partitioned Regions](#) on page 213

### Default Attributes

Region Attributes

data-policy:	PERSISTENT-PARTITION
--------------	----------------------

### gfsh Command Example

```
gfsh>create region --name=myPPregion --type=PARTITION_PERSISTENT
Member | Status
----- | -----
server1 | Region "/myPPregion" created on "server1"
server2 | Region "/myPPregion" created on "server2"

gfsh>describe region --name=myPPregion
Name : myPPregion
Data Policy : persistent partition
Hosting Members : server1
                  server2

Non-Default Attributes Shared By Hosting Members

Type | Name | Value
---- | ---- | -----
Region | size | 0
```

### cache.xml Example

```
<region name="myPPregion">
    <region-attributes
        data-policy="persistent-partition" />
</region>
```

## PARTITION\_PERSISTENT\_OVERFLOW

### Description

A region configured with the PARTITION\_PERSISTENT\_OVERFLOW has local state that is partitioned across each peer member that created the region. The region writes its state to disk and can recover that state when the member restarts. . The region removes the values of entries from memory when it detects that the JVM is running low on memory.

For more information, see:

- [RegionShortcuts for Peers and Servers](#) on page 151
- [Persistence and Overflow](#) on page 245
- [Partitioned Regions](#) on page 213

### Default Attributes

Region Attributes

data-policy:	PERSISTENT-PARTITION
--------------	----------------------

Eviction Attributes

eviction-algorithm:	lru-heap-percentage
eviction-action:	overflow-to-disk

### gfsh Command Example

```
gfsh>create region --name=myPPOregion --type=PARTITION_PERSISTENT_OVERFLOW
Member | Status
-----+-----
server2 | Region "/myPPOregion" created on "server2"
server1 | Region "/myPPOregion" created on "server1"

gfsh>describe region --name=myPPOregion
.....
Name : myPPOregion
Data Policy : persistent partition
Hosting Members : server1
                  server2
```

Non-Default Attributes Shared By Hosting Members

Type	Name	Value
Region	size	0
Eviction	eviction-algorithm	lru-heap-percentage
	eviction-action	overflow-to-disk

### cache.xml Example

```
<region name="myPPOregion">
  <region-attributes>
    <data-policy="persistent-partition">
      <eviction-attributes>
        <lru-heap-percentage
          action="overflow-to-disk" />
```

```

        </eviction-attributes>
    </region-attributes>
</region>
```

## PARTITION\_PROXY

### Description

A region configured with the PARTITION\_PROXY region shortcut has no local state and forwards all operations to a peer region configured with the PARTITION shortcut or a peer region configured with the PARTITION\_PERSISTENT shortcut.

See:

- [RegionShortcuts for Peers and Servers](#) on page 151
- [Partitioned Regions](#) on page 213

### Default Attributes

Region Attributes

data-policy:	PARTITION
--------------	-----------

Partition Attributes

local-max-memory:	0
-------------------	---

### gfsh Command Example

```

gfsh>create region --name=myPPXYregion --type=PARTITION_PROXY
Member | Status
----- | -----
server1 | Region "/myPPXYregion" created on "server1"
server2 | Region "/myPPXYregion" created on "server2"

gfsh>describe region --name=myPPXYregion
...
Name : myPPXYregion
Data Policy : partition
Accessor Members : server1
                  server2

Non-Default Attributes Shared By Accessor Members

      Type |       Name |   Value
----- | ----- | -----
Region | size | 0
Partition | local-max-memory | 0
```

### cache.xml Example

```

<region name="myPPXYregion">
    <region-attributes
        data-policy="partition">
        <partition-attributes
            local-max-memory="0" />
    </region-attributes>
</region>
```

## PARTITION\_PROXY\_REDUNDANT

### Description

A region configured with the PARTITION\_PROXY\_REDUNDANT region shortcut has no local state and forwards all operations to a peer region configured with the PARTITION\_REDUNDANT shortcut or a peer region configured with the PARTITION\_REDUNDANT\_PERSISTENT shortcut.

See:

- [RegionShortcuts for Peers and Servers](#) on page 151
- [Partitioned Regions](#) on page 213

### Default Attributes

Region Attributes

data-policy:	PARTITION
--------------	-----------

Partition Attributes

redundant-copies:	1
local-max-memory	0

### gfsh Command Example

```
gfsh>create region --name=myPPRregion --type=PARTITION_PROXY_REDUNDANT
Member | Status
----- | -----
server1 | Region "/myPPRregion" created on "server1"
server2 | Region "/myPPRregion" created on "server2"
```

```
gfsh>describe region --name=myPPRregion
.....
```

```
Name : myPPRregion
Data Policy : partition
Accessor Members : server1
                  server2
```

Non-Default Attributes Shared By Accessor Members

Type	Name	Value
Region	size	0
Partition	redundant-copies	1
	local-max-memory	0

### cache.xml Example

```
<region name="myPPRregion">
  <region-attributes
    data-policy="partition">
    <partition-attributes
      redundant-copies="1"
      local-max-memory="0" />
  </region-attributes>
</region>
```

## PARTITION\_REDUNDANT

### Description

A region configured with the PARTITION\_REDUNDANT region shortcut has local state that is partitioned across each peer member that created the region. The region maintains an extra copy of the data in memory.

See:

- [RegionShortcuts for Peers and Servers](#) on page 151
- [Partitioned Regions](#) on page 213

### Default Attributes

Region Attributes

data-policy:	PARTITION
--------------	-----------

Partition Attributes

redundant-copies:	1
-------------------	---

### gfsh Command Example

```
gfsh>create region --name=myPRregion --type=PARTITION_REDUNDANT
Member | Status
----- | -----
server1 | Region "/myPRregion" created on "server1"
server2 | Region "/myPRregion" created on "server2"

gfsh>describe region --name=myPRregion
.....
Name : myPRregion
Data Policy : partition
Hosting Members : server1
                  server2
```

Non-Default Attributes Shared By Hosting Members

Type	Name	Value
Region	size	0
Partition	redundant-copies	1

### cache.xml Example

```
<region name="myPRregion">
  <region-attributes
    data-policy="partition">
    <partition-attributes
      redundant-copies="1"/>
  </region-attributes>
</region>
```

## PARTITION\_REDUNDANT\_HEAP\_LRU

### Description

A region configured with the PARTITION\_REDUNDANT\_HEAP\_LRU region shortcut has local state that is partitioned across each peer member that created the region. The region keeps an extra copy of the data in memory. The region destroys entries when it detects that the JVM is running low on memory.

For more information, see:

- [Persistence and Overflow](#) on page 245
- [Local Regions](#) on page 208
- [Eviction](#) on page 248
- [RegionShortcuts for Peers and Servers](#) on page 151
- [Partitioned Regions](#) on page 213

### Default Attributes

Region Attributes

data-policy:	PARTITION
--------------	-----------

Eviction Attributes

eviction-algorithm:	lru-heap-percentage
eviction-action:	local-destroy

Partition Attributes

redundant-copies:	1
-------------------	---

### gfsh Command Example

```
gfsh>create region --name=myPRHLregion --type=PARTITION_REDUNDANT_HEAP_LRU
Member | Status
----- | -----
server1 | Region "/myPRHLregion" created on "server1"
server2 | Region "/myPRHLregion" created on "server2"

gfsh>describe region --name=myPRHLregion
Name : myPRHLregion
Data Policy : partition
Hosting Members : server1
                  server2
```

### Non-Default Attributes Shared By Hosting Members

Type	Name	Value
Region	size	0
Eviction	eviction-algorithm	lru-heap-percentage
	eviction-action	local-destroy
Partition	redundant-copies	1

### cache.xml Example

```
<region name="myPRHLregion">
    <region-attributes
        data-policy="partition">
        <partition-attributes redundant-copies="1"/>
        <eviction-attributes>
            <lru-heap-percentage action="local-destroy">
                <class-name>
                    com.gemstone.gemfire.internal.size.SizeClassOnceObjectSizer
                </class-name>
            </lru-heap-percentage>
        </eviction-attributes>
    </region-attributes>
</region>
```

## PARTITION\_REDUNDANT\_OVERFLOW

### Description

A region configured with the PARTITION\_REDUNDANT\_OVERFLOW region shortcut has local state that is partitioned across each peer member that created the region. The region keeps an extra copy of the data in memory. The region moves the values of entries to disk when it detects that the JVM is running low on memory.

For more information, see:

- [Persistence and Overflow](#) on page 245
- [Local Regions](#) on page 208
- [Eviction](#) on page 248
- [RegionShortcuts for Peers and Servers](#) on page 151
- [Partitioned Regions](#) on page 213

### Default Attributes

Region Attributes

data-policy:	PARTITION
--------------	-----------

Eviction Attributes

eviction-algorithm:	lru-heap-percentage
eviction-action:	overflow-to-disk

Partition Attributes

redundant-copies:	1
-------------------	---

### gfsh Command Example

```
gfsh>create region --name=myPROregion --type=PARTITION_REDUNDANT_OVERFLOW
Member | Status
-----
server2 | Region "/myPROregion" created on "server2"
server1 | Region "/myPROregion" created on "server1"

gfsh>describe region --name=myPROregion
.....
```

```

Name : myPROregion
Data Policy : partition
Hosting Members : server1
                server2

Non-Default Attributes Shared By Hosting Members

Type | Name | Value
-----|-----|-----
Region | size | 0
Eviction | eviction-algorithm | lru-heap-percentage
           | eviction-action | overflow-to-disk
Partition | redundant-copies | 1

```

### **cache.xml Example**

```

<region name="myPROregion">
  <region-attributes>
    <data-policy="partition">
      <partition-attributes>
        redundant-copies="1" />
      <eviction-attributes>
        <lru-heap-percentage
          action="overflow-to-disk" />
      </eviction-attributes>
    </partition-attributes>
  </region-attributes>
</region>

```

## **PARTITION\_REDUNDANT\_PERSISTENT**

### **Description**

A region configured with the PARTITION\_REDUNDANT\_PERSISTENT region shortcut has local state that is partitioned across each peer member that created the region. The region writes its state to disk and recovers its state from disk when the region is created. The region maintains an extra copy of the data in memory.

For more information, see:

- [Persistence and Overflow](#) on page 245
- [RegionShortcuts for Peers and Servers](#) on page 151.
- [Partitioned Regions](#) on page 213

### **Default Attributes**

Region Attributes

data-policy:	PERSISTENT-PARTITION
--------------	----------------------

Partition Attributes

redundant-copies:	1
-------------------	---

### **gfsh Command Example**

```

gfsh>create region --name=myPRPregion --type=PARTITION_REDUNDANT_PERSISTENT
Member | Status
-----|-----
server2 | Region "/myPRPregion" created on "server2"

```

```
server1 | Region "/myPRPregion" created on "server1"

gfsh>describe region --name=myPRPregion
-----
Name : myPRPregion
Data Policy : persistent partition
Hosting Members : server1
                server2

Non-Default Attributes Shared By Hosting Members

  Type      |      Name      |  Value
-----|-----|-----
Region   | size        | 0
Partition | redundant-copies | 1
```

### **cache.xml Example**

```
<region name="myPRPregion">
  <region-attributes
    data-policy="persistent-partition">
    <partition-attributes
      redundant-copies="1"/>
  </region-attributes>
</region>
```

## **PARTITION\_REDUNDANT\_PERSISTENT\_OVERFLOW**

### **Description**

A region configured with the PARTITION\_REDUNDANT\_PERSISTENT\_OVERFLOW region shortcut has local state that is partitioned across each peer member that created the region. The region writes its state to disk and recovers its state from disk when the region is created. The region maintains an extra copy of the data in memory. The region removes the values of entries from memory when it detects that the JVM is running out of memory.

For more information, see:

- [Persistence and Overflow](#) on page 245
- [RegionShortcuts for Peers and Servers](#) on page 151.
- [Partitioned Regions](#) on page 213

### **Default Attributes**

#### Region Attributes

data-policy:	PERSISTENT-PARTITION
--------------	----------------------

#### Eviction Attributes

eviction-algorithm:	lru-heap-percentage
eviction-action:	overflow-to-disk

#### Partition Attributes

redundant-copies:	1
-------------------	---

**gfsh Command Example**

```
create region --name=myPRPOregion
--type=PARTITION_REDUNDANT_PERSISTENT_OVERFLOW

describe region --name=myPRPOregion
.....
Name : myPRPOregion
Data Policy : persistent partition
Hosting Members : server1
                server2

Non-Default Attributes Shared By Hosting Members

      Type      |      Name      |      Value
-----+-----+-----+-----+
Region | size | 0
Eviction | eviction-algorithm | lru-heap-percentage
          | eviction-action | overflow-to-disk
Partition | redundant-copies | 1
```

**cache.xml Example**

```
<region name="myPRPOregion">
<region-attributes
    data-policy="persistent-partition">
    <partition-attributes
        redundant-copies="1"/>
    <eviction-attributes>
        <lru-heap-percentage
            action="overflow-to-disk"/>
    </eviction-attributes>
</region-attributes>
</region>
```

**REPLICATE****Description**

A region configured with the REPLICATE region shortcut has local state that is kept in sync with all other peer regions configured with a REPLICATE data policy.

For more information, see:

- [Persistence and Overflow](#) on page 245
- [RegionShortcuts for Peers and Servers](#) on page 151
- [Distributed and Replicated Regions](#) on page 233

**Default Attributes**

Region Attributes

data-policy:	REPLICATE
scope:	distributed-ack

**gfsh Command Example**

```
gfsh>create region --name=myRregion --type=REPLICATE
```

REPLICATE	REPLICATE_PERSISTENT
REPLICATE_OVERFLOW	
REPLICATE_PERSISTENT_OVERFLOW	REPLICATE_HEAP_LRU
REPLICATE_PROXY	

```
gfsh>create region --name=myRegion --type=REPLICATE
Member | Status
----- | -----
server1 | Region "/myRegion" created on "server1"
server2 | Region "/myRegion" created on "server2"
```

### cache.xml Example

```
<region name="myRegion">
    <region-attributes
        scope="distributed-ack"
        data-policy="replicate" />
</region>
```

## REPLICATE\_HEAP\_LRU

### Description

A region configured with the REPLICATE\_HEAP\_LRU region shortcut has local state that is kept in sync with all other peer regions configured with a REPLICATE data policy. The region destroys entries when it detects that the JVM is running low on memory.



**Note:** Normally, you cannot have a `local-destroy` eviction policy on replicated regions. However, the REPLICATE\_HEAP\_LRU region is not the same as other replicated regions. This region is preloaded and remains in synch by registering interest in the keys of its peer's replicated regions.

For more information, see:

- [RegionShortcuts for Peers and Servers](#) on page 151
- [Eviction](#) on page 248
- [Distributed and Replicated Regions](#) on page 233

### Default Attributes

#### Region Attributes

data-policy:	preloaded
scope:	distributed-ack

#### Eviction Attributes

eviction-algorithm:	lru-heap-percentage
eviction-action:	local-destroy



**Note:** Normally, you cannot have a `local-destroy` eviction policy on replicated regions. However, the REPLICATE\_HEAP\_LRU region is not the same as other replicated regions. This region is preloaded and remains in synch by registering interest in the keys of its peer's replicated regions.

## Subscription Attributes

interest-policy:	all
------------------	-----

### gfsh Command Example

```
gfsh>create region --name=myRHLregion --type=REPLICATE_HEAP_LRU
Member | Status
----- | -----
server1 | Region "/myRHLregion2" created on "server1"
server2 | Region "/myRHLregion2" created on "server2"

gfsh>describe region --name=myRHLregion
-----
Name : myRHLregion
Data Policy : preloaded
Hosting Members : server1
                server2

Non-Default Attributes Shared By Hosting Members

      Type | Name | Value
----- | ----- | -----
Region | size | 0
Eviction | eviction-algorithm | lru-heap-percentage
           | eviction-action | local-destroy
```

### cache.xml Example

```
<region name="myRHLregion">
  <region-attributes
    scope="distributed-ack"
    data-policy="preloaded">
    <subscription-attributes
      interest-policy="all"/>
    <eviction-attributes>
      <lru-heap-percentage
        action="local-destroy">
    </lru-heap-percentage>
    </eviction-attributes>
  </region-attributes>
</region>
```

## REPLICATE\_OVERFLOW

### Description

A region configured with the REPLICATE\_OVERFLOW region shortcut has local state that is kept in sync with all other peer regions configured with a REPLICATE data policy.

For more information, see:

- [Persistence and Overflow](#) on page 245
- [RegionShortcuts for Peers and Servers](#) on page 151
- [Distributed and Replicated Regions](#) on page 233

## Default Attributes

### Region Attributes

data-policy:	replicate
scope:	distributed-ack

### Eviction Attributes

eviction-algorithm:	lru-heap-percentage
eviction-action:	overflow-to-disk

## gfsh Command Example

```
gfsh>create region --name=myROregion --type=REPLICATE_OVERFLOW
Member | Status
----- | -----
server2 | Region "/myROregion" created on "server2"
server1 | Region "/myROregion" created on "server1"

gfsh>describe region --name=myROregion
Name : myROregion
Data Policy : replicate
Hosting Members : server1
                  server2

Non-Default Attributes Shared By Hosting Members

Type | Name | Value
----- | ----- | -----
Region | size | 0
Eviction | eviction-algorithm | lru-heap-percentage
           | eviction-action | overflow-to-disk
```

## cache.xml Example

```
<region name="myROregion">
  <region-attributes
    scope="distributed-ack"
    data-policy="replicate">
    <eviction-attributes>
      <lru-heap-percentage
        action="overflow-to-disk"/>
    </eviction-attributes>
  </region-attributes>
</region>
```

## REPLICATE\_PERSISTENT

### Description

A region configured with the REPLICATE\_PERSISTENT region shortcut has local state that is kept in sync with all other peer regions that are configured with a REPLICATE data policy. The region writes its state to disk and recovers that state when the member restarts.

For more information, see:

- [Persistence and Overflow](#) on page 245
- [RegionShortcuts for Peers and Servers](#) on page 151
- [Distributed and Replicated Regions](#) on page 233

## Default Attributes

Region Attributes

data-policy:	PERSISTENT-REPLICATE
scope:	distributed-ack

## gfsh Command Example

```
gfsh>create region --name=myRPregion --type=REPLICATE_PERSISTENT
Member | Status
----- | -----
server2 | Region "/myRPregion" created on "server2"
server1 | Region "/myRPregion" created on "server1"

gfsh>describe region --name=myRPregion
.....
Name : myRPregion
Data Policy : persistent replicate
Hosting Members : server1
                  server2

Non-Default Attributes Shared By Hosting Members

Type | Name | Value
---- | --- | ---
Region | size | 0
```

## cache.xml Example

```
<region name="myRPregion">
  <region-attributes
    scope="distributed-ack"
    data-policy="empty"/>
</region>
```

## REPLICATE\_PERSISTENT\_OVERFLOW

### Description

A region configured with the REPLICATE\_PERSISTENT\_OVERFLOW region shortcut has local state that is kept in sync with all other peer regions configured with a REPLICATE data policy. The region writes its state to disk and recovers that state when the member restarts. The region removes the values of entries from memory when it detects that the JVM is running low on memory.

For more information, see:

- [Persistence and Overflow](#) on page 245
- [RegionShortcuts for Peers and Servers](#) on page 151
- [Distributed and Replicated Regions](#) on page 233

## Default Attributes

### Region Attributes

data-policy:	PERSISTENT-REPLICATE
scope:	distributed-ack

### Eviction Attributes

eviction-algorithm:	lru-heap-percentage
eviction-action:	overflow-to-disk

## gfsh Command Example

```
gfsh>create region --name=myRPOregion --type=REPLICATE_PERSISTENT
REPLICATE_PERSISTENT           REPLICATE_PERSISTENT_OVERFLOW

gfsh>create region --name=myRPOregion --type=REPLICATE_PERSISTENT_OVERFLOW
Member   | Status
----- | -----
server1 | Region "/myRPOregion" created on "server1"
server2 | Region "/myRPOregion" created on "server2"

gfsh>describe region --name=myRPOregion
Name          : myRPOregion
Data Policy    : persistent replicate
Hosting Members : server1
                  server2
```

### Non-Default Attributes Shared By Hosting Members

Type	Name	Value
Region	size	0
Eviction	eviction-algorithm	lru-heap-percentage
	eviction-action	overflow-to-disk

## cache.xml Example

```
<region name="myRPOregion">
  <region-attributes
    scope="distributed-ack"
    data-policy="persistent-replicate">
    <eviction-attributes>
      <lru-heap-percentage
        action="overflow-to-disk" />
    </eviction-attributes>
  </region-attributes>
</region>
```

## REPLICATE\_PROXY

### Description

A region configured with the REPLICATE\_PROXY region shortcut has no local state and forwards all operations to a peer region configured with the [REPLICATE](#) or [REPLICATE\\_PERSISTENT](#) region shortcut.

For more information, see:

- [RegionShortcuts for Peers and Servers](#) on page 151
- [Distributed and Replicated Regions](#) on page 233

### Default Attributes

Region Attributes

data-policy:	EMPTY
scope:	distributed-ack

### gfsh Command Example

```
gfsh>create region --name=myRPXYregion --type=REPLICATE_PROXY
Member | Status
----- | -----
server1 | Region "/myRPXYregion" created on "server1"
server2 | Region "/myRPXYregion" created on "server2"

gfsh>describe region --name=myRPXYregion
Name : myRPXYregion
Data Policy : empty
Accessor Members : server1
                  server2
```

Non-Default Attributes Shared By Accessor Members

Type	Name	Value
Region	size	0

### cache.xml Example

```
<region name="myRPXYregion">
    <region-attributes
        scope="distributed-ack"
        data-policy="empty"/>
</region>
```

# Chapter 67

---

## Exceptions and System Failures

Your application needs to catch certain classes to handle all the exceptions and system failures thrown by vFabric GemFire.

- **GemFireCheckedException** . This class is the abstract superclass of exceptions that are thrown and declared. Wherever possible, GemFire exceptions are checked exceptions. GemFireCheckedException is a GemFire version of `java.lang.Exception`.
- **GemFireException** . This class is the abstract superclass of unchecked exceptions that are thrown to indicate conditions for which the developer should not normally need to check. You can look at the subclasses of `GemFireException` to see all the runtime exceptions in the GemFire system; see the class hierarchy in the online Java API documentation. `GemFireException` is a GemFire version of `java.lang.RuntimeException`. You can also look at the method details in the Region API javadocs for GemFire exceptions you may want to catch.
- **SystemFailure** . In addition to exception management, GemFire provides a class to help you manage catastrophic failure in your distributed system, particularly in your application. The Javadocs for this class provide extensive guidance for managing failures in your system and your application. See `SystemFailure` in the `com.gemstone.gemfire` package.

To see the exceptions thrown by a specific method, refer to the method's online Java documentation.

A GemFire system member can also throw exceptions generated by third-party software such as JGroups or `java.lang` classes. For assistance in handling these exceptions, see the vendor documentation.



# Chapter 68

---

## Memory Requirements for Cached Data

GemFire solutions architects need to estimate resource requirements for meeting application performance, scalability and availability goals.

These requirements include estimates for the following resources:

- memory
- number of machines
- network bandwidth

The information here is only a guideline, and assumes a basic understanding of GemFire. While no two applications or use cases are exactly alike, the information here should be a solid starting point, based on real-world experience. Much like with physical database design, ultimately the right configuration and physical topology for deployment is based on the performance requirements, application data access characteristics, and resource constraints (i.e., memory, CPU, and network bandwidth) of the operating environment.

- [Core Guidelines for GemFire Data Region Design](#) on page 989
- [Memory Usage Overview](#) on page 989
- [Cache Overhead](#) on page 990
- [Sockets](#) on page 991

### Core Guidelines for GemFire Data Region Design

- For 32-bit JVMs: If you have a small data set (< 2GB) and a read-heavy requirement, you should be using replicated regions.
- For 64-bit JVMs: If you have a data set that is larger than 50-60% of the JVM heap space you can use replicated regions. For read heavy applications this can be a performance win. For write heavy applications you should use partitioned caches.
- If you have a large data set and you are concerned about scalability you should be using partitioned regions.
- If you have a large data set and can tolerate an on-disk subset of data, you should be using either replicated regions or partitioned regions with overflow to disk.
- If you have different data sets that meet the above conditions, then you might want to consider a hybrid solution mixing replicated and partition regions. Do not exceed 50 to 75% of the JVM heap size depending on how write intensive your application is.

### Memory Usage Overview

The following guidelines should provide a rough estimate of the amount of memory consumed by your system. A worksheet is available to help calculate your capacity using this information.

Memory calculation about keys and entries (objects) and region overhead for them can be divided by the number of members of the distributed system for data placed in partitioned regions only. For other regions, the calculation is for

each member that hosts the region. Memory used by sockets, threads, and the small amount of application overhead for GemFire is per member.

For each entry added to a region, the GemFire cache API consumes a certain amount of memory to store and manage the data. This overhead is required even when an entry is overflowed or persisted to disk. Thus objects on disk take up some JVM memory, even when they are paged to disk. The Java cache overhead introduced by a region, using a 32-bit JVM, 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 JVMs, the usage will usually be larger than with 32-bit JVMs. As much as 80% more memory may be required for 64-bit JVMs, due in large part to the fact that all address and integers are 64 bits, not 32 bits. The companion spread sheet does provide for 64-bit JVMs but it is necessarily an approximation due to the platform and JVM issues mentioned above.



**Note:** Objects in GemFire 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. See Data Serialization [Data Serialization](#) on page 257.

There are several additional considerations for calculating your memory requirements:

- **Size of your stored data.** Use the larger of the serialized and deserialized forms for each data object type. For `DataSerializable` object classes, the non-serialized form will generally be the larger of the two.

Objects in GemFire are serialized for storage into partitioned regions and for all distribution activities, including moving data to disk for overflow and persistence. For optimum performance, GemFire tries to reduce the number of times an object is serialized and deserialized, so your objects may be stored in serialized or non-serialized form in the cache.

- **Application object overhead for your data.** These are the estimated values for 32-bit JVMs and 64-bit JVMs. Sizes may vary slightly between JVMs and platforms.

- Object header. On 32-bit JVMs, 12 bytes. (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.) On 64-bit JVMs, 20 bytes. 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 JVMs, 8 bytes for fields of type `double` or `long`, 4 bytes per field for all others. On 64-bit JVMs, the size is the same as 32-bit except for fields that are references to objects, which take 8 bytes.

## Cache Overhead

This table gives estimates for the cache overhead in a 32-bit JVM. The overhead is required even when an entry is overflowed or persisted to disk. Actual memory use varies based on a number of factors, including the JVM type and the platform you run on. For 64-bit JVMs, the usage will usually be larger than with 32-bit JVMs and may be as much as 80% more.

When calculating cache overhead...	You should add...
For each region	81 bytes per entry

**Note:** Memory consumption for object headers and object references can vary for 64-bit JVMs, different JVM implementations, and different JDK versions.

When calculating cache overhead...	You should add...
And concurrency checking is enabled (enabled by default)	16 bytes per entry  (See <a href="#">Overhead for Consistency Checks</a> on page 241.)
And statistics are enabled for the member	16 bytes per entry
And the region is partitioned	16 bytes per entry
And the region is persisted and/or overflowed	40 bytes per entry
And the region has an LRU eviction controller	16 bytes per entry
And the region has global scope	90 bytes per entry
And the region has entry expiration configured	147 bytes per entry
For each optional user attribute	52 bytes 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 indexed expression, 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 is reached if 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 the indexed expression, but no two region entries have the same value for it, then the index introduces at most  $236 C + 75$  bytes per region entry, where  $C$  is the average number of values per region entry for the expression.

## Sockets

Servers always maintain two outgoing connections to each of their peers. So for each peer a server has, there are four total connections: two going out to the peer and two coming in from the peer.

The server threads that service client requests also communicate with peers to distribute events and forward client requests. If the server's GemFire connection property `conserve-sockets` is set to true (the default), these threads use the already-established peer connections for this communication.

If `conserve-sockets` is false, each thread that services clients establishes two of its own individual connections to its server peers, one to send, and one to receive. Each socket uses a file descriptor, so the number of available sockets is governed by two operating system settings:

- maximum open files allowed on the system as a whole
- maximum open files allowed for each session

In servers with many threads servicing clients, if `conserve-sockets` is set to false, the demand for connections can easily overrun the number of available sockets. Even with `conserve-sockets` set to false, you can cap the number of these connections by setting the server's `max-threads` parameter.

Since each client connection takes one server socket on a thread to handle the connection, and since that server acts as a proxy on partitioned regions to get results, or execute the function service on behalf of the client, for partitioned regions, if `conserve-sockets` is set to false, this also results in a new socket on the server being opened to each peer. Thus  $N$  sockets are opened, where  $N$  is the number of peers. Large number of clients simultaneously connecting to a large set of peers with a partitioned region with `conserve-sockets` set to false can cause a huge amount of memory to be consumed by socket. Set `conserve-socket` to true in these instances.



**Note:** There is also JVM overhead for the thread stack for each client connection being processed, set at 256KB or 512 KB for most JVMs . On some JVMs you can reduce it to 128KB. You can use the GemFire `max-threads`

property or the GemFire max-connections property to limit the number of client threads and thus both thread overhead and socket overhead.

The following table lists the memory requirements based on connections.

<b>Connections</b>	<b>Memory requirements</b>
Per socket	32,768 /socket (configurable) Default value per socket should be set to a number > 100 + sizeof (largest object in region) + sizeof (largest key)
If server (for example if there are clients that connect to it)	= (lesser of max-threads property on server or max-connections)* (socket buffer size +thread overhead for the JVM )
Per member of the distributed system if conserve sockets is set to true	4* number of peers
Per member, if conserve sockets is set to false	4 * number of peers hosting that region* number of threads
If member hosts a Partitioned Region, If conserve sockets set to false and it is a Server (this is cumulative with the above)	=< max-threads * 2 * number of peers  <b>Note:</b> it is = 2* current number of clients connected * number of peers. Each connection spawns a thread.
<b>Subscription Queues</b>	
Per Server, depending on whether you limit the queue size. If you do, you can specify the number of megabytes or the number of entries until the queue overflows to disk. When possible, entries on the queue are references to minimize memory impact. The queue consumes memory not only for the key and the entry but also for the client ID/thread ID as well as for the operation type. Since you can limit the queue to 1 MB, this number is completely configurable and thus there is no simple formula.	1 MB +
<b>GemFire classes and JVM overhead</b>	Roughly 50MB
<b>Thread overhead</b>	
Each concurrent client connection into the a server results in a thread being spawned up to max-threads setting. After that a thread services multiple clients up to max-clients setting.	There is a thread stack overhead per connection (at a minimum 256KB to 512 KB, you can set it to smaller to 128KB on many JVMs.)

## Additional Resources

In addition to the guidelines described in this topic, see the following resources:

- [GemFire Memory Sizing Guidelines](#)
- [Data Sizer Java Utility Class](#)

# Glossary

This glossary defines terms used in vFabric GemFire documentation.

## **ACK wait threshold**

A time-to-wait for message acknowledgment between system members.

## **administrative event**

See [event](#) on page 997.

## **API**

Application Programming Interface. GemFire provides APIs to cached data for Java applications.

## **Application Cache Node**

One of two vFabric GemFire licenses you can obtain for custom evaluation or production. Application Cache Node is the vFabric Suite offering for vFabric GemFire. See [Choosing a License Option Based on Topology](#) on page 46.

## **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](#) on page 1003.

## **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](#) on page 1006.

## **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](#) on page 1000 and [remote](#) on page 1003.

---

**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 [event](#) on page 997.

**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](#) on page 994.

**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](#) on page 1001 and [netLoad](#) on page 1001.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 that can service client connections.

**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](#) on page 999 and [global transaction](#) on page 998.

**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](#) on page 994 and [netWrite](#) on page 1001.

**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](#) on page 1003.

## **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 sender events going to remote distributed systems can all be conflated.

## **connection**

The connection used by an application to access a GemFire 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. Gateway senders connect to a remote gateway receiver 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](#) on page 1002.

## **coordinator**

The member of the distributed system that sends out membership views. This is typically the locator in GemFire.

## **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 store](#) on page 996.

### **data entry**

See [entry](#) on page 997.

### **data fabric**

Also referred to as an in-memory data grid or an enterprise data fabric. vFabric GemFire is an implementation of a data fabric. A data fabric is a distributed, memory-based data management platform that uses cluster-wide resources -- memory, CPU, network bandwidth, and optionally local disk -- to manage application data and application logic (behavior). The data fabric uses dynamic replication and data partitioning techniques to offer continuous availability, very high performance, and linear scalability for data intensive applications, all without compromising on data consistency even when exposed to failure conditions.

### **Data Management Node**

One of two vFabric GemFire licenses you can obtain for custom evaluation or production. For details on configurations that require a Data Management Node license and its associated upgrades, see [Choosing a License Option Based on Topology](#) on page 46.

### **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.

### **data region (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 store**

In the context of a region, a member configured to store data for the region. This is used mostly for partitioned regions, where data is spread across the distributed system among the data stores. See also [data accessor](#) on page 995.

### **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 sender 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 cache events with each other, forming a single, logical system.

## **distributed-ack scope**

Data distribution setting that causes synchronous distribution operations, which wait for acknowledgment 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](#) on page 1003, [entry key](#) on page 997, and [entry value](#) on page 997.

## **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 receiver**

A gateway receiver defines connection information for receiving region events that were distributed from a gateway sender in a multi-site deployment.

## **gateway sender**

A gateway sender defines a single remote distributed system site and an associated queue for distributing region events in a multi-site deployment.

## **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, and logging management.

## **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](#) on page 999 and [cache transaction](#) on page 994.

---

## **HTTP**

World Wide Web's Hypertext Transfer Protocol. 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](#) on page 1006 and [expiration action](#) on page 998.

### **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**

Java DataBase Connectivity. A programming interface that lets Java applications access a database via the SQL language.

## **JMX**

Java Management eXtensions. A set of specifications for dynamic application and network management in the J2EE development and application environment.

## **JNDI**

Java Naming and Directory Interface. 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**

Java Transaction API. 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](#) on page 998.

## **JVM**

Java Virtual Machine. A virtual machine capable of handling Java bytecode.

---

## **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](#) on page 1003.

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](#) on page 998.

## **LRU**

Least recently used. 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](#) on page 1004.

## **machine**

Any GemFire-supported physical machine or VMware Virtual Machine.

## **member**

A process that has defined a connection to a GemFire distributed system and created a GemFire cache. This can be a Java or Native Client application. This can also be a GemFire process such as a locator or cacheserver. The minimal GemFire process configuration is a single member that is connected to a distributed system.

---

## **message queue**

A first-in, first-out data structure in a GemFire 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](#) on page 995.

## **mirroring**

See [replicate](#) on page 1004.

## **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 refids. 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 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 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 with a data-policy of empty, normal and preloaded.

## **netWrite**

The method used by GemFire 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**

Object Query Language, 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.

---

## **partition**

The memory in each member that is reserved for a specific partitioned region's use.

## **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 vFabric 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-receivers or gateway-senders 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](#) on page 995.

## **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](#) on page 1003.

## **pure Java mode**

Running GemFire without the use of the GemFire native library. GemFire 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](#) on page 1002.

## **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](#) on page 995.

## **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](#) on page 1005 and [value-index](#) on page 1007.

## **region**

A logical grouping of data within a cache. Regions usually contain data entries (see [entry](#) on page 997). 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](#) on page 997.

## **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](#) on page 1000.

### **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 JVM's tenured garbage collection (GC) to control heap use and protect your JVM 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](#) on page 1000 and [eviction-attributes](#) on page 997.

### **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](#) on page 1004 and [Data Management Node](#) on page 996.

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**

Structured Query Language.

**SSL**

Secure Socket Layer. A protocol for secure communication between Java VMs.

**standalone distributed system**

A distributed system configured for no communication with peers. Client applications are generally defined with standalone distributed systems, so there is no peer communication and all event and data communication is done between the client member and the server tier.

**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](#) on page 1003 and [value-index](#) on page 1007.

**system member**

See [machine](#) on page 1000.

**TCP**

The Transmission Control Protocol 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](#) on page 1006.

---

**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](#) on page 993.

**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](#) on page 999 and [expiration action](#) on page 998.

**transaction**

See [cache transaction](#) on page 994 and [global transaction](#) on page 998.

**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](#) on page 1006.

**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](#) on page 1006.

**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 system.

**TTL**

See [time-to-live](#) on page 1006.

**UDP**

The User Datagram Protocol 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](#) on page 1005.

**unicast**

A message sent from one process to another process (point-to-point communications). Both UDP and TCP provide unicast messaging.

**URI**

Uniform Resource Identifier. 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 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](#) on page 1005 and [range-index](#) on page 1003.

### **view**

A collection of member identifiers that defines the membership group in JGroups.

### **Virtual Machine**

A completely isolated operating system installation within your normal operating system. This is generally implemented by software emulation or hardware virtualization.

### **VMware virtual machine**

Also referred to as a VMware VM. A VMware VM is a tightly isolated software container that can run its own operating systems and applications as if it were a physical computer. A VMware VM behaves exactly like a physical computer and contains its own virtual (ie, software-based) CPU, RAM hard disk and network interface card (NIC). An operating system can't tell the difference between a VMware VM and a physical machine, nor can applications or other computers on a network. Even the VMware VM thinks it is a "real" computer. Nevertheless, a VMware VM is composed entirely of software and contains no hardware components whatsoever.

### **XML**

EXtensible Markup Language. 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.