Tanzu GemFire

Tanzu GemFire 10.2

Contents

Documentation

Tanzu GemFire 10.2

API Reference Documentation

Documentation of Related Products

Release Notes

New in Tanzu GemFire 10.2

Enhanced WAN Copy Efficiency with Wildcard Region Selection

WAN Copy Improvements

Message Loss Detection and New Stats

GemFire messaging now uses connection pooling

Enhanced Failure Handling and Retry Support for WAN Replication

Improved Redaction

OS-Level Statistics Migrated to OSHI

Cluster Health Documentation

Tanzu GemFire on vSphere reference architecture documentation

Off platform WAN access

Improved TLS 1.3 handshake performance

Improved logging when a member is forcibly disconnected

New statistics

HTTP authentication

Create and delete initializers using gfsh

Increase the maximum socket lease time to 7 days

ResourceManager updates and support for Generational ZGC

Improvements to gfsh deadlock detection Command

Accounting for JVM pauses has been improved

Consistent path naming in GemFire tgz bundles

Default Setting Changes

Deprecations

Known Issues

Resolved Issues

Security fixes

v10.2.0 Security Fixes

Supported Configurations and System Requirements

Tanzu GemFire Supported Platforms

Host Machine Requirements

Java Support

How JDK 17 Encapsulation Affects Tanzu GemFire

Access Required by Tanzu GemFire

Access to the Types Used in Application Data

Additional Access Required by All Tanzu GemFire Processes

Granting Access to Encapsulated Types

Opening Specific Packages
Using an Argument File
Opening All JDK Packages
Java Garbage Collection Defaults and Recommendations
Enable or Disable ZGC/Generational ZGC using gfsh

Supported Platforms and System Requirements for Tools

Pulse System Requirements

VSD System Requirements

Getting Started with Tanzu GemFire

About Tanzu GemFire
Main Concepts and Components

Main Features
High Read-and-Write Throughput
Low and Predictable Latency
High Scalability
Continuous Availability
Reliable Event Notifications
Parallelized Application Behavior on Data Stores
Shared-Nothing Disk Persistence
Reduced Cost of Ownership
Single-Hop Capability for Client/Server
Client/Server Security
Multisite Data Distribution
Continuous Querying
Heterogeneous Data Sharing

Installing Tanzu GemFire
Obtaining and Installing Security Updates

Installing Tanzu GemFire from a Compressed TAR File
Prerequisites
Procedure

Installing Tanzu GemFire Extensions

Uninstalling Tanzu GemFire

Upgrading Tanzu GemFire
Upgrade Details

Planning an Upgrade
Guidelines for Upgrading
Version Compatibilities
Upgrade to a Newer Version

Rolling Upgrade
Rolling Upgrade Limitations and Requirements
Rolling Upgrade Guidelines
Rolling Upgrade Procedure
Upgrade Locators
Upgrade Servers
Upgrade Clients

Offline Upgrade
- Offline Upgrade Guidelines
- Offline Upgrade Procedure

Upgrading Clients
- Remove or Replace Obsolete Identifiers
- Protocol Alignment

Upgrading from 9.x to 10.2
- Procedure

Tanzu GemFire in 15 Minutes or Less
- Step 1. Install Tanzu GemFire
- Step 2. Use gfsh to start a locator
- Step 3. Start Pulse
- Step 4. Start a server
- Step 5. Create a replicated, persistent region
- Step 6. Manipulate data in the region and demonstrate persistence
- Step 7. Examine the effects of replication
- Step 8. Restart the cache servers in parallel
- Step 9. Shut down the system including your locators
- Step 10. What to do next…

Add GemFire to Your Project
- Java and Spring
  - Prerequisites
  - Adding Tanzu GemFire to a Maven Project
  - Adding Tanzu GemFire to a Gradle Project
  - Java Client Example
  - Spring Boot for GemFire
    - Spring Boot Client Example for a Maven Project
    - Spring Boot Client Example for a Gradle Project
  - Confirm Dependency Usage
- .NET Framework Native Client
  - Example: Put, Get, and Remove with VMware Tanzu GemFire Native .NET Client (C#)
- C++ Native Client
  - Example: Put, Get, and Remove with VMware Tanzu GemFire Native C++ Client

Configuring and Running a Cluster

Overview of the Cluster Configuration Service
- Why Use the Cluster Configuration Service
- Using the Cluster Configuration Service
- How the Cluster Configuration Service Works
- gfsh Commands that Create Cluster Configurations
- gfsh Limitations
- Deactivating the Cluster Configuration Service

Tutorial—Creating and Using a Cluster Configuration

Deploying Application JARs to Tanzu GemFire Members
- About Deploying JAR Files and the Cluster Configuration Service
- Automatic Function Registration
- Naming JAR Files
- Deployment Location for JAR Files
- Versioning of JAR Files

Automatic Class Path Loading

Classloader Isolation

Disable Classloader Isolation

gfsh

Locators

Servers

Deploying Application JARs Using the Chained Approach

Deployment Location for JAR Files (chained model)

Versioning of JAR Files (chained model)

Automatic Class Path Loading (chained model)

Using Member Groups

Exporting and Importing Cluster Configurations

Exporting a Cluster Configuration

Importing a Cluster Configuration

Cluster Configuration Files and Troubleshooting

Troubleshooting Tips

Sizing a GemFire Cluster

Overview

Resource Considerations

Experimentation and Testing

Requirements and Assumptions

Architectural and Design Considerations

Serialization

Per-entry Memory Overhead

Partitioned Region Scalability

Redundancy

Relationship Between Horizontal and Vertical Scale

NUMA Considerations

GemFire Queues

Sizing Process

Step 1: Domain object sizing

Step 2: Estimating total memory and system requirements

Step 3: Vertical Sizing

Locator Sizing

Notes on GC

Step 4: Scale-out Validation

Step 5: Projection to Full Scale

Sizing Quick Reference

Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS

Deploying Configuration Files without the Cluster Configuration Service

Main Steps to Deploying Configuration Files

Tanzu GemFire Configuration Files

Default File Specifications and Search Locations

Changing the File Specifications

Deploying Configuration Files in JAR Files

Starting Up and Shutting Down Your System

Starting Up Your System
Starting Up After Losing Data on Disk
Shutting Down the System
Using the shutdown Command
Shutting Down System Members Individually
Option for System Member Shutdown Behavior

Running GemFire Locator Processes

Start a Locator with gfsh
Locator Configuration and Log Files
Locators and the Cluster Configuration Service
Check Locator Status
Stop the Locator
Locators and Multi-Site (WAN) Deployments
Start a Locator with Custom Shell Script
Start a Locator Programmatically

Running GemFire Server Processes

Start a Server with gfsh
Default Server Configuration and Log Files
Check Server Status
Stop a Server
Start a Server with Custom Shell Script
Start a Server Programmatically

Managing System Output Files

Firewall Considerations

Firewalls and Connections

Firewalls and Ports

Limiting Ephemeral Ports for Peer-to-Peer Membership
Properties for Firewall and Port Configuration
Default Port Configurations
Properties for Firewall and Port Configuration in Multi-Site (WAN) Configurations

Basic Configuration and Programming

Distributed System and Cache Configuration

Cluster Members

Member Overview
Membership and System Topologies
Multi-site Installations

Setting Properties

Options for Configuring the Cache and Data Regions

Local and Remote Membership and Caching

Cache Management

Introduction to Cache Management

The Caching APIs
The Cache XML
Create and Close a Cache
Export and Import a Cache Snapshot

Cache Management with gfsh and the Cluster Configuration Service
Managing a Peer or Server Cache
Managing a Client Cache
Managing a Cache in a Secure System
Managing RegionServices for Multiple Secure Users
Requirements and Caveats for RegionService
Launching an Application after Initializing the Cache
Implementing an Initializer Class
Configuring an Initializer
Configuring an Initializer in cache.xml
Configure and Manage Initializers using gfsh
Data Regions
Region Management
Creating a Region
Creating a Region with gfsh
Creating a Region Through the cache.xml File
cache.xml File Examples
Creating a Region Through the API
API Examples
Create and Access Data Subregions
Update the Configuration of Data Regions
Invalidate a Region
Clear a Region
Destroy a Region
Close a Region
Region Naming
Region Shortcuts and Custom Named Region Attributes
Shortcut Attribute Options
RegionShortcuts for Peers and Servers
ClientRegionShortcuts for Clients
Storing and Retrieving Region Shortcuts and Custom Named Region Attributes
Examples
Managing Region Attributes
Define Region Attributes
Modify Region Attributes
Creating Custom Attributes for Regions and Entries
Limitations and Alternatives
Building a New Region with Existing Content
Data Entries
Managing Data Entries
Keys
Create and Update Entries
The getAll Operation
The putAll Operation

The removeAll Operation
Retrieving Region Entries from Proxy Members
Using gfsh to get and put

Copy on Read Behavior

Requirements for Using Custom Classes in Data Caching
Classloader
Class Requirements
Classes Used as Keys

Topologies and Communication

Topology and Communication General Concepts

Topology Types
Peer-to-Peer Configuration
Client/Server Configuration
Multi-site Configuration

Planning Topology and Communication
Determine Protocols and Addresses
Set Up Membership and Communication

How Member Discovery Works
Peer Member Discovery
Standalone Member
Client Discovery of Servers
Multi-site Discovery

How Communication Works

How Multi-site (WAN) Systems Work

Using Bind Addresses
Peer and Server Communication
Gateway Receiver Communication
Locator Communication

Choosing Between IPv4 and IPv6

Peer-to-Peer Configuration
Configure Peer-to-Peer Discovery
Allow Peer-to-Peer Discovery
Disallow Peer-to-Peer Discovery
Organize Peers into Logical Member Groups

Client/Server Configuration

Standard Client/Server Deployment

How Server Discovery Works
Basic Configuration
Using Member Groups

How Client/Server Connections Work
Pool connections
How the Pool Chooses a Server Connection
How the Pool Connects to a Server
How the Pool Manages Pool Connections

Subscription connections
How the Pool Manages Subscription Connections
How the Pool Conditions Server Load

Configuring a Client/Server System

Organizing Servers Into Logical Member Groups

Client/Server Example Configurations

Examples of Standard Client/Server Configuration
Example—Standalone Publisher Client, Client Pool, and Region
Example—Standalone Subscriber Client
Example of a Static Server List in Client/Server Configuration

Fine-Tuning Your Client/Server Configuration

How Server Load Conditioning Works

Multi-site (WAN) Configuration

Overview of Multi-site Caching

Consistency for WAN Updates
Discovery for Multi-Site Systems
Gateway Senders
Serial Gateway Senders
Parallel Gateway Senders
Gateway Sender Queues
Timing of Connections
High Availability for Gateway Senders
Stopping Gateway Senders
Pausing Gateway Senders
Gateway Receivers

Multi-site (WAN) Topologies

Fully Connected Mesh Topology
Ring Topology
Hybrid Multi-site Topology
Unsupported Topologies

Configuring a Multi-site (WAN) System

Prerequisites
Main Steps
Configure Gateway Senders
Create Data Regions for Multi-site Communication
Configure Gateway Receivers
Accessing a Site Behind an SNI Proxy

Filtering Events for Multi-Site (WAN) Distribution

Configuring Multi-Site Event Filters

Resolving Conflicting Events

Implementing a GatewayConflictResolver

Handling WAN Replication Errors

Callback API
GatewaySender property
Gateway Sender gfsh Example
Gateway Sender cache.xml Example

GatewayReceiver Properties
Gateway Receiver gfsh Example
gateway receiver cache.xml Example
Updated exception handling for exceptions from the gateway receiver
Storing gateway event contents in the user-provided region

Managing Tanzu GemFire

Tanzu GemFire Management and Monitoring

Management and Monitoring Features
References

Overview of GemFire Management and Monitoring Tools
gfsh Command-line tool
Executing gfsh commands with the management API
Member Configuration Management
Java Management Extension (JMX) MBeans
Tanzu GemFire Java API
Tanzu GemFire Management Console
Tanzu GemFire Pulse
JConsole
Prometheus Metrics Endpoint
Tanzu Observability

Prometheus Metrics Endpoint
Configure Tanzu GemFire Metrics
Configure GemFire
Example Startup
Security

Tanzu Observability
Telegraf and Tanzu Observability Proxy Setup Example
Install Telegraf and the Proxy
Configure Telegraf
Verification and Troubleshooting
Default Dashboard Metrics
Tanzu GemFire Cluster Metrics
Tanzu GemFire Features Metrics
WAN Gateway Metrics

Architecture and Components
Architecture
Managed Node
JMX Manager Node
JMX Integration
Management APIs
Tanzu GemFire Management and Monitoring Tools

Starting a JMX Manager
Configuring a JMX Manager
Stopping a JMX Manager

JMX Manager Operations

Federated MBean Architecture
Federation of Tanzu GemFire MBeans and MBeanServers

MBean Proxy Naming Conventions
Use of MXBeans
MBean Proxy Creation

List of GemFire JMX MBeans

JMX Manager MBeans
ManagerMXBean
DistributedSystemMXBean
DistributedRegionMXBean
DistributedLockServiceMXBean
Managed Node MBeans
MemberMXBean
CacheServerMXBean
RegionMXBean
LockServiceMXBean
DiskStoreMXBean
AsyncEventQueueMXBean
LocatorMXBean
LuceneServiceMXBean
GatewaySenderMXBean
GatewayReceiverMXBean

Browsing GemFire MBeans through JConsole

GemFire JMX MBean Notifications

Notification Federation
Attaching Listeners to MXBeans
System Alert Notifications

List of JMX MBean Notifications
MemberMXBean Notifications
MemberMXBean Gateway Notifications
CacheServerMXBean Notifications
DistributedSystemMXBean Notifications

Configuring RMI Registry Ports and RMI Connectors
Configuring JMX Manager Port and Bind Addresses
Using Out-of-the-Box RMI Connectors

Executing gfsh Commands through the Management API

Managing Heap and Off-heap Memory

Managing Heap Memory
Using the GemFire Resource Manager
How Background Eviction Is Performed
Heap Size
Concurrent Mark-Sweep Garbage Collector (CMS) Heaps
Z Garbage Collector (ZGC) Heaps
Other ZGC Guidance
Controlling Heap Use with the Resource Manager
Configure Tanzu GemFire for Heap LRU Management
Tuning the JVM’s Garbage Collection Parameters
Concurrent Mark-Sweep (CMS) Garbage Collector
Garbage First (G1) Garbage Collector

Z Garbage Collector (ZGC)
Monitor and Tune Heap LRU Configurations
Example Scenario
Resource Manager Example Configurations

Heap LRU Eviction With G1GC

Configuring G1GC for GemFire on JDK17
Managing Humongous Objects
Tuning Frequent Garbage Collection to Support Heap LRU Eviction
Troubleshooting Heap LRU Eviction: Waking the Garbage Collector When Your Application is Idle
Analyzing GC and Heap Behavior
Fixing the Problem with -XX:G1PeriodicGCInterval
Troubleshooting Heap LRU Eviction: Ensuring The Garbage Collector Responds Quickly to Sudden Application Activity
Response to Sudden Application Activity
Setting -XX:G1PeriodicGCInterval

Managing Off-Heap Memory

On-heap and Off-heap Objects
Off-heap Recommendations
Implementation Details
Controlling Off-heap Use with the Resource Manager
Specifying Off-heap Memory
gfsh Off-heap Support
ResourceManager API
Tuning Off-heap Memory Usage

Locking Memory (Linux Systems Only)

Disk Storage

How Disk Stores Work

What Tanzu GemFire Writes to the Disk Store
Disk Store State

Disk Store File Names and Extensions

File Names
File Extensions

Disk Store Operation Logs

When Disk Store Oplogs Reach the Configured Disk Capacity

Configuring Disk Stores

Defining and Configuring Disk Stores

Design Your Disk Stores
Create and Configure Your Disk Stores
Modifying Disk Stores
Configuring Regions, Queues, and PDX Serialization to Use the Disk Stores
Configuring Disk Stores on Gateway Senders
Segmented Disk Stores
Disabling Segmented Disk Stores

Disk Store Configuration Parameters

Disk Store Configuration Attributes and Elements
disk-dirs Element

Modifying the Default Disk Store
Change the Behavior of the Default Disk Store
Optimizing a System with Disk Stores
Start Up and Shut Down with Disk Stores
Start Up
Start Up Procedure
Example Startup to Illustrate Ordering
Shutdown
Disk Store Management
Disk Store Management Commands and Operations
Online Disk Store Operations
Offline Disk Store Operations
Validating a Disk Store
Running Compaction on Disk Store Log Files
Log File Compaction for the Online Disk Store
Run Online Compaction
Run Offline Compaction
Performance Benefits of Manual Compaction
Directory Size Limits
Example Compaction Run
Keeping a Disk Store Synchronized with the Cache
Change Region Configuration
Take a Region Out of Your Cache Configuration and Disk Store
Configuring Disk Free Space Monitoring
Handling Missing Disk Stores
Show Missing Disk Stores
Revoke Missing Disk Stores
Altering When Buffers Are Flushed to Disk
Modifying Disk Flushes for the Operating System
Modifying Tanzu GemFire to Flush Buffers on Disk Writes
Creating Backups for System Recovery and Operational Management
Making a Backup While the System is Online
What a Full Online Backup Saves
What an Incremental Online Backup Saves
Disk Store Backup Directory Structure and Contents
Offline Members—Manual Catch-Up to an Online Backup
Restore Using a Backup Made While the System Was Online
Cache and Region Snapshots
Usage and Performance Notes
Cache Consistency and Concurrent Operations
Performance Considerations
Exporting Cache and Region Snapshots
Exporting Cache Snapshots
Exporting a Region Snapshot
Export Example with Options

Importing Cache and Region Snapshots
Import Requirements
Import Limitations
Importing Cache Snapshots
Importing a Region Snapshot
Filtering Entries During Import or Export
Reading Snapshots Programmatically
Region Compression
What Gets Compressed
Guidelines to Using Compression
How to Enable Compression in a Region
How to Check Whether Compression is Enabled
Working with Compressors
Changing the Compressor for an Already Compressed Region
Comparing Performance of Compressed and Non-Compressed Regions
Monitoring Compression Performance
Network Partitioning
How Network Partitioning Management Works
Failure Detection and Membership Views
Failure Detection
Membership Views
Membership Coordinators, Lead Members, and Member Weighting
Membership Coordinators and Lead Members
Member Weighting System
Sample Member Weight Calculations
Network Partitioning Scenarios
What the Losing Side Does
What Isolated Members Do
Configure Tanzu GemFire to Handle Network Partitioning
Preventing Network Partitions
Security
Security Implementation Introduction and Overview
Security Features
Overview
Hardening Guide
Hardening Recommendations
Using OQL Method Authorizers
Communication Ports
Resources That Must Be Protected
Log File Locations
Where to Place Security Configuration Settings
Enable Security with Property Definitions
security-manager Property
Apply security-manager to All Members
Is Cluster Management Enabled?

Apply security-manager to Non-participating Servers
Callbacks
security-post-processor Property

Authentication

Implementing Authentication

How Authentication Works
How a Server Sets Its Credentials
Add Settings to the Server Properties File
Implement the AuthInitialize Interface for the Server
How a Client Cache Sets its Credentials
Implement the AuthInitialize Interface for the Client
Provide Secure Access to Client Credentials
How Other Components Set Their Credentials
Implement SecurityManager Interface

Authentication Example

Username and Password Authentication Example
Token with Expiry Authentication Example

Implementing Authentication Expiry

Authentication Expiry Considerations

Authorization

Implementing Authorization

How Authorization Works
Resource Permissions
Implement Authorization
Authorization of Function Execution
Authorization of Methods Invoked from Queries

Method Invocation Authorizers

Overview
GemFire Authorizers
RestrictedMethodAuthorizer
UnrestrictedMethodAuthorizer
JavaBeanAccessorMethodAuthorizer
RegExMethodAuthorizer
Custom Authorizers
How Authorization Works
Implementing a Method Authorizer
Changing the Method Authorizer

Authorization Example

Disclaimer
User Authorization Example
Method Invocation Authorization Example
Token with Expiry Authorization Example

Custom Security Manager Example

Create a List of Approved Users
Create a BasicSecurityManager
Authentication
Authorization

Starting a GemFire Cluster with the Security Manager
Create a Java Client application
Create a Region on the GemFire Cluster
Set Credential Properties
Client Authorization Error

Post Processing of Region Data
Implement Post Processing

SSL/TLS

Configuring SSL/TLS
SSL-Configurable Components
SSL Configuration Properties
Example: secure communications throughout
Example: non-secure cluster communications, secure client/server
SSL Property Reference Tables
Procedure

SSL/TLS Sample Implementation
Provider-Specific Configuration File
gemfire.properties File
gfsecurity.properties File
Locator Startup
Other Member Startup
Connecting to a Running Cluster

Performance Tuning and Configuration

Improving Performance on vSphere
Operating System Guidelines
NUMA, CPU, and BIOS Settings
Physical and Virtual NIC Settings
VMware vSphere vMotion and DRS Cluster Usage
VMware vSphere Snapshots
Placement and Organization of Virtual Machines
Virtual Machine Memory Reservation
vSphere High Availability and VMware Tanzu GemFire
Storage Guidelines

Performance Controls

Data Serialization

Setting Cache Timeouts

Controlling Socket Use

Management of Slow Receivers

Increasing the Ratio of Cache Hits

File System

System Member Performance

Member Properties

JVM Memory Settings and System Performance

Garbage Collection and System Performance

Optimizing Garbage Collection
Using Off-heap Memory

GemFire Health Checklist

JVM Pauses High
Detecting JVM Pauses
Impact
Where to Look
Analyzing sampleTime Variation
Remedies

CPU Steal High
Impact
Where to Look
Remedies for High CPU Steal

Resource Manager Health Factors
Impact
Where to Look
Determining Historical Heap States from ResourceManager Statistics
Remedies

Reply Timeouts High
Impact
Where to Look
Remedies
Lower GemFire Versions
More information

Partitioned Region Redundancy Low
Impact
Redundancy Conditions
Where to Look
Remedies

Disk Low
Impact
Where to Look
Remedies

Slow Receivers with TCP/IP

Preventing Slow Receivers

Managing Slow Receivers

Slow distributed-ack Messages

Socket Communication

Setting Socket Buffer Sizes

Ephemeral TCP Port Limits

Making Sure You Have Enough Sockets
Socket Sharing
Socket Lease Time
Calculating Connection Requirements
Peer-to-Peer Socket Requirements Per Member

Server Socket Requirements Per Server
Client Socket Requirements per Client

TCP/IP KeepAlive Configuration

TCP/IP Peer-to-Peer Handshake Timeouts

Configuring Sockets in Multi-Site (WAN) Deployments
Multi-site (WAN) Socket Requirements
Member produces SocketTimeoutException

Maintaining Cache Consistency
General Guidelines
Guidelines for Multi-Site Deployments

Deactivating TCP SYN Cookies

Logging

How GemFire Logging Works
Custom Appenders in Tanzu GemFire
Advanced Configuration

Understanding Log Messages and Their Categories
Structure of a Log Message
Log File Name
How the System Renames Logs
Log Level

Naming, Searching, and Creating Log Files
Log File Naming Recommendation
Searching the Log Files
Creating Your Own Log Messages

Configuring Logging
log-level
log-file
log-file-size-limit
log-disk-space-limit

Configuring Log4j 2 for Tanzu GemFire
Using Different Front-End Logging APIs to Log to Log4j2
Customizing Your Own log4j2.xml File

Statistics

How Statistics Work

Transient Region and Entry Statistics

Application-Defined and Custom Statistics

Configuring and Using Statistics
Configuring Cluster or Server Statistics
Configure Transient Region and Entry Statistics
Configure Custom Statistics
Controlling the Size of Archive Files

Viewing Archived Statistics

Troubleshooting and System Recovery

Producing Artifacts for Troubleshooting

Diagnosing System Problems

- Locator does not start
- Application or cache server process does not start
- Application or cache server does not join the cluster
- Member process seems to hang
- Member process does not read settings from the gemfire.properties file
- Cache creation fails - must match schema definition root
- Cache is not configured properly
- Unexpected results for keySetOnServer and containsKeyOnServer
- Data operation returns PartitionOfflineException
- Entries are not being evicted or expired as expected
- Cannot find the log file
- OutOfMemoryError
- PartitionedRegionDistributionException
- PartitionedRegionStorageException
- Application crashes without producing an exception
- Timeout alert
- Member produces SocketTimeoutException
- Member logs ForcedDisconnectException, Cache and DistributedSystem forcibly closed
- Members cannot see each other
- One part of the cluster cannot see another part
- Data distribution has stopped, although member processes are running
- Distributed-ack operations take a very long time to complete
- Slow system performance
- cannot get Windows performance data
- Java applications on 64-bit platforms hang or use 100% CPU

System Failure and Recovery

- Planning for Data Recovery
- Network Partitioning, Slow Response, and Member Removal Alerts
  - Network Partitioning Detected
  - Member Taking Too Long to Respond
  - No Locators Can Be Found
  - Warning Notifications Before Removal
  - Member Is Forced Out
  - Disk Access Exceptions
- How Data is Recovered From Persistent Regions

Handling Forced Cache Disconnection Using Auto-reconnect

- How the Auto-reconnection Process Works
- Managing the Auto-reconnection Process
- Operator Intervention

Recovering from Application and Cache Server Crashes

Recovering from Crashes with a Peer-to-Peer Configuration

- Recovery for Partitioned Regions
- Recovery for Distributed Regions
- Recovery for Regions of Local Scope
- Recovering Data from Disk

Recovering from Crashes with a Client/Server Configuration

Recovering from Server Failure
Recovering from Client Failure

Recovering from Machine Crashes

Recovery Procedure
Data Recovery for Partitioned Regions
Data Recovery for Distributed Regions
Data Recovery in a Client/Server Configuration

Recovering from ConfictingPersistentDataExceptions

Independently Created Copies
Starting New Members First
A Network Failure Occurs and Network Partitioning Detection is Deactivated
Salvaging Data

Preventing and Recovering from Disk Full Errors

Recovering from Disk Full Errors

Understanding and Recovering from Network Outages

What Happens During a Network Outage
Recovery Procedure
Effect of Network Failure on Partitioned Regions
Effect of Network Failure on Distributed Regions
Effect of Network Failure on Persistent Regions
Effect of Network Failure on Client/Server Installations

Log Messages and Solutions

above heap eviction threshold
below heap eviction threshold
above heap critical threshold
below heap critical threshold
Query execution canceled after exceeding max execution time
Query execution canceled due to memory threshold crossed in system
Thread warnings
Disconnecting old DistributedSystem to prepare for a reconnect attempt
Attempting to reconnect to the DistributedSystem. This is attempt #n
Unable to form a TCP/IP connection in a reasonable amount of time
Received Suspect Message
Seconds Have Elapsed
Message Loss Detected
Member is not responding to heartbeat requests
Enabled-network-partition-detection is set to false
Statistics sampling thread detected a wakeup delay
Redundancy has dropped below configured copies
Rejected connection
PCC service metrics component failing to connect to locator/server
SSLHandshakeException: is disabled
Unable To Create New Native Thread
Available file descriptors in the system: XX is below the set threshold: 50
An IOException has occurred due to exhaustion of file descriptors. The cache will be closed.
CommitConflictException
Initialization of Region <_B__RegionName_BucketNumber> Completed
Unknown pdx Type error
Error calculating expiration

PdxType limitations for GFSH queries
Apache.Geode.Client.AllConnectionsInUseException
org.apache.geode.pdx.PdxInitializationException
Format of the string «cache xml file's content» used for parameterization is unresolvable
RegionExistException
Missing Diskstore Exception
Could not create an instance of a class
PartitionedRegion#cleanupFailedInitialization: Failed to clean the PartitionRegion allPartitionedRegions
Could not find any server to create primary client queue on.
Cluster configuration service not available
The secondary map already contained an event from hub null so ignoring new event
Create is present in more than one Oplog. This should not be possible. The Oplog Key ID for this entry is
Detected conflicting PDX types during import
A tenured heap garbage collection has occurred
Allocating larger network read buffer
Socket send buffer size is instead of the requested
quorum has been lost
possible loss of quorum due to the loss of cache processes
Membership service failure: Exiting due to possible network partition event due to loss of <n> cache processes
had a weight of
An additional Function Execution Processor thread is being launched
Sending new view
Received new view
Admitting member
Member at unexpectedly left the distributed cache
Cache server: failed accepting client connection
Remote host closed connection during handshake
SSL peer shut down incorrectly
Function: cannot be executed because the members [list of members] are running low on memory
Region bucket has persistent data that is no longer online stored at these locations
Region <regionName> has potentially stale data. Buckets [list] are waiting for another offline member
Cluster membership failed due to ForcedDisconnectException

Developing with Tanzu GemFire

Region Data Storage and Distribution

Storage and Distribution Options

Peer-to-Peer Region Storage and Distribution
Storing Data in the Local Cache

Region Types

Partitioned Regions
Replicated Regions
Distributed, Non-Replicated Regions
Local Regions

Region Data Policies

Region Data Stores and Data Accessors

Partitioned Regions

Understanding Partitioning
- Data Partitioning
- Partitioned Region Operation
- Additional Information About Partitioned Regions

Configuring Partitioned Regions

Configuring the Number of Buckets for a Partitioned Region
- Calculate the Total Number of Buckets for a Partitioned Region

Custom-Partitioning and Colocating Data

Understanding Custom Partitioning and Data Colocation
- Custom Partitioning
- Data Colocation Between Regions

Custom-Partition Your Region Data

Standard Custom Partitioning

Fixed Custom Partitioning

Colocate Data from Different Partitioned Regions

Configuring High Availability for Partitioned Regions

Understanding High Availability for Partitioned Regions
- Controlling Where Your Primaries and Secondaries Reside
- Running Processes in Virtual Machines
- Reads and Writes in Highly-Available Partitioned Regions

Configure High Availability for a Partitioned Region

Set the Number of Redundant Copies

Configure Redundancy Zones for Members

Set Enforce Unique Host

Configure Member Crash Redundancy Recovery for a Partitioned Region

Configure Member Join Redundancy Recovery for a Partitioned Region

Configuring Single-Hop Client Access to Partitioned Regions
- Understanding Client Single-Hop Access to Partitioned Regions
  - How Single-Hop and Connection Limit Settings Interact
  - How Single-Hop and Connection Timeout Settings Interact
- Optimizing Single-Hop Access
  - Balancing Single-Hop Server Connection Use
  - Performance Tuning Considerations
- Connection Limit Adaptation Formula

Rebalancing Partitioned Region Data
- How Partitioned Region Rebalancing Works
- When to Rebalance a Partitioned Region
- How to Simulate Region Rebalancing
- Automated Rebalancing

Checking Redundancy in Partitioned Regions

Restoring Redundancy in Partitioned Regions
Moving Partitioned Region Data to Another Member
Distributed and Replicated Regions
How Distribution Works
Options for Region Distribution
How Replication and Preloading Work
Initialization of Replicated and Preloaded Regions
Behavior of Replicated and Preloaded Regions After Initialization
Configure Distributed, Replicated, and Preloaded Regions
Local Destroy and Invalidate in the Replicated Region
Locking in Global Regions
Lock Timeouts
Optimize Locking Performance
Examples
Consistency for Region Updates
Consistency Checking by Region Type
Partitioned Region Consistency
Replicated Region Consistency
Non-Replicated Regions and Client Cache Consistency
Configuring Consistency Checking
Overhead for Consistency Checks
How Consistency Checking Works for Replicated Regions
How Destroy and Clear Operations Are Resolved
About Region.clear() Operations
Transactions with Consistent Regions
How Consistency Is Achieved in WAN Deployments
General Region Data Management
Persistence and Overflow
How Persistence and Overflow Work
How Data Is Persisted and Overflowed
Persistence
Overflow
Persistence and Overflow Together
Persistence and Multi-Site Configurations
Configure Region Persistence and Overflow
Overflow Configuration Examples
Eviction
How Eviction Works
Eviction Actions
Eviction in Partitioned Regions
Configure Data Eviction
Region-based eviction criteria (Entry count, bytes used)
Application heap memory usage-based eviction
Examples

Expiration

How Expiration Works

- **Expiration Types**
- **Expiration Actions**
- **Entry Expiration in Replicated Regions and Partitioned Regions**
- **Interaction Between Expiration Settings and netSearch**

Configure Data Expiration

- **Types of Expiration**
- **Enable Statistics**
- **Configure Expiration Attributes**
  - Available Attributes
  - Example (cache.xml)
  - Example (gfsh)
- **Special Considerations**
  - Distributed Expiration Behavior
  - Partitioned Regions
- **Custom Expiration**
  - Example Java Class
  - Example (cache.xml)
  - Example (gfsh)
- **Performance Tuning for Expiration**

Keeping the Cache Synchronized with Outside Data Sources

Overview of Outside Data Sources

Configuring Database Connections Using JNDI

- **Example DataSource Configurations in cache.xml**
- **XAPooledDataSource cache.xml Example (Derby)**
- **JNDI Binding Configuration Properties for Different XAPooledDataSource Connections**
- **ManagedDataSource Connection Example (Derby)**
- **PooledDataSource Example (Derby)**
- **SimpleDataSource Connection Example (Derby)**

How Data Loaders Work

- **Data Loading in Partitioned Regions**
- **Data Loading in Distributed Regions**
- **Data Loading in Local Regions**

Implement a Data Loader

- **Implement the CacheLoader Interface**
- **Configure and Deploy**
- **Implementing a Server or Peer with a Cache Loader**

Data Serialization

Overview of Data Serialization

- **Data Serialization Options**
- **Differences between Tanzu GemFire Serialization (PDX or Data Serializable) and Java Serialization**

GemFire PDX Serialization

GemFire PDX Serialization Features

- **Application Versioning of PDX Domain Objects**

Portability of PDX Serializable Objects
Reduced Deserialization of Serialized Objects

High Level Steps for Using PDX Serialization
PDX and Multi-Site (WAN) Deployments

PDX Serialization Best Practices
Field Compatibility
PDX Type Persistence
Backup and Recovery
Client-Side PDX Handling
WAN Configuration

Using Automatic Reflection-Based PDX Serialization

Customizing Serialization with Class Pattern Strings

Extending the ReflectionBasedAutoSerializer
Reasons to Extend the ReflectionBasedAutoSerializer
Overriding ReflectionBasedAutoSerializer Behavior
Example of Optimizing Autoserialization of BigInteger and BigDecimal Types

Serializing Your Domain Object with a PdxSerializer

Implementing PdxSerializable in Your Domain Object

Programming Your Application to Use PdxInstances

Adding JSON Documents to the GemFire Cache
Overview
Usage
BSON
PDX

Using PdxInstanceFactory to Create PdxInstances
Enum Objects as PdxInstances

Persisting PDX Metadata to Disk

Using PDX Objects as Region Entry Keys

GemFire Data Serialization (DataSerializable and DataSerializer)
Data Serialization with the DataSerializable Interface
Serializing Your Domain Object with DataSerializer

Standard Java Serialization

Events and Event Handling

How Events Work
Events Features
Types of Events
Event Cycle
Event Objects
Event Distribution
Event Handlers and Region Data Storage
Multiple Listeners
Event Ordering

Peer-to-Peer Event Distribution
Events in a Partitioned Region

Events in a Distributed Region
Managing Events in Multi-threaded Applications

Client-to-Server Event Distribution

Server-to-Client Event Distribution
Server-to-Client Message Tracking
Client Interest Registration on the Server
Server Failover

Multi-Site (WAN) Event Distribution

Queuing Events for Distribution
Operation Distribution from a Gateway Sender
How a Gateway Sender Processes Its Queue
How a Gateway Sender Handles Batch Processing Failure

List of Event Handlers and Events

Event Handlers
Events

Implementing GemFire Event Handlers

Implementing Cache Event Handlers

Installing Multiple Listeners on a Region

Implementing an AsyncEventListener for Write-Behind Cache Event Handling

How an AsyncEventListener Works
Operation Distribution from an AsyncEventQueue
Guidelines for Using an AsyncEventListener
Implementing an AsyncEventListener
Processing AsyncEvents
Configuring an AsyncEventListener

How to Safely Modify the Cache from an Event Handler Callback

Operations to Avoid in Event Handlers
How to Perform Distributed Operations Based on Events

Cache Event Handler Examples

Declaring and Loading an Event Handler with Parameters
Installing an Event Handler Through the API
Installing Multiple Listeners on a Region
Installing a Write-Behind Cache Listener

Configuring Peer-to-Peer Event Messaging

Configuring Client/Server Event Messaging

Configuring Highly Available Servers

Highly Available Client/Server Event Messaging

Change Server Queue Synchronization Frequency
Set Frequency of Orphan Removal from the Secondary Queues

Implementing Durable Client/Server Messaging

Configure the Client as Durable
Configure Durable Subscriptions and Continuous Queries
Program the Client to Manage Durable Messaging
Initial Operation
Disconnection
Reconnection

Durable Event Replay
Application Operations During Interest Registration
Tuning Client/Server Event Messaging
Conflate the Server Subscription Queue
Limit the Server's Subscription Queue Memory Use
Tune the Client's Subscription Message Tracking Timeout
Configuring Multi-Site (WAN) Event Queues
Persisting an Event Queue
Configuring Persistence for Async Event Queues
Configuring Dispatcher Threads and Order Policy for Event Distribution
Using Multiple Dispatcher Threads to Process a Queue
Performance and Memory Considerations
Configuring the Ordering Policy for Serial Queues
Examples—Configuring Dispatcher Threads and Ordering Policy for a Serial Gateway Sender Queue
Conflating Events in a Queue
Examples—Configuring Conflation for a Gateway Sender Queue
Delta Propagation
How Delta Propagation Works
General Characteristics of Delta Propagation
Supported Topologies and Limitations
When to Avoid Delta Propagation
Delta Propagation Properties
delta-propagation
cloning-enabled
Implementing Delta Propagation
Errors In Delta Propagation
Delta Propagation Example
Querying
Querying FAQ and Examples
How do I write and execute a query against a Tanzu GemFire region?
Can I see query string examples, listed by query type?
Which APIs should I use to write my queries?
How do I invoke an object’s method in a query?
Can I invoke a static method on an object in a query?
How do I write a reusable query?
When should I create indexes to use in my queries?
How do I create an index?
Can I create indexes on overflow regions?
Can I query a partitioned region? Can I perform a join query on a partitioned region?
How can I improve the performance of a partitioned region query?
Which query language elements are supported in Tanzu GemFire?
How do I debug queries?
Can I use implicit attributes or methods in my query?

Can I instruct the query engine to use specific indexes with my queries?
How do I perform a case-insensitive search on a field in OQL?

Querying with OQL
Advantages of OQL

Writing and Executing a Query in OQL
Querying a Local Cache
Querying a Server Cache from a Client
Next Steps

Building a Query String

IMPORT Statement

FROM Clause
Path Expressions
Aliases and Synonyms
Object Typing

WHERE Clause
Implementing equals and hashCode Methods
Querying Serialized Objects
Attribute Visibility
Joins
LIKE
Case Insensitive Fields
Method Invocations
- Methods without parameters
- Methods with parameters
- Methods calls with the SecurityManager enabled

Enum Objects
IN and SET
Double.NaN and Float.NaN Comparisons
Arithmetic Operations

SELECT Statement
SELECT Statement Results
DISTINCT
LIMIT
ORDER BY
Preset Query Functions

OQL Aggregate Functions
GROUP BY
MIN
MAX
COUNT
SUM
AVG

OQL Syntax and Semantics

Supported Character Sets

Supported Keywords

Case Sensitivity

Comments in Query Strings

Query Language Grammar

- **Language Grammar**
  - Notation used in the grammar:
  - Grammar rules:
- **Language Notes**

Operators

- **Comparison Operators**
- **Logical Operators**
- **Unary Operators**
- **Arithmetic Operators**
- **Map and Index Operators**
- **Dot, Right Arrow, and Forward Slash Operators**

Reserved Words

- **Reserved Words**

Supported Literals

- **The Difference Between NULL and UNDEFINED**
- **Comparing Values With java.util.Date**
- **Type Conversion**
- **Binary Numeric Promotion**
- **Method Invocation Conversion**
- **Temporal Type Conversion**
- **Enum Conversion**
- **Query Evaulation of Float.NaN and Double.NaN**

Query Language Restrictions and Unsupported Features

Advanced Querying

Querying a Specific Member

Performance Considerations

Managing Queries

- **List Queries**
- **Describe Queries**
- **Cancel Queries**

Monitoring Low Memory When Querying

- **Partitioned Region Queries and Low Memory**

Timeouts for Long-Running Queries

Using Query Bind Parameters

- **Sample Code**
- **Using Query Bind Parameters in the Path Expression**

Querying Partitioned Regions

Using ORDER BY on Partitioned Regions

Querying a Partitioned Region on a Single Node

Optimizing Queries on Data Partitioned by a Key or Field Value

Performing an Equi-Join Query on Partitioned Regions

Partitioned Region Query Restrictions

Query Restrictions in Partitioned Regions

Query Debugging

Working with Indexes

Tips and Guidelines for Using Indexes

Tips for Writing Queries that Use Indexes

Creating, Listing, and Removing Indexes

Creating Indexes

Listing Indexes

Removing Indexes

Creating Key Indexes

Examples of Creating a Key Index

Creating Hash Indexes

Hash Index Performance

Performance Considerations

Limitations

Examples of Creating a Hash Index

Creating Indexes on Map Fields (“Map Indexes”)

Creating Multiple Indexes Concurrently

Maintaining Indexes (Synchronously or Asynchronously) and Index Storage

Index Maintenance Behavior

Internal Index Structure and Storage

Using Query Index Hints

Using Indexes on Single Region Queries

Using Indexes with Equi-Join Queries

Using Indexes with Overflow Regions

Using Indexes on Equi-Join Queries using Multiple Regions

Index Samples

Continuous Querying

How Continuous Querying Works

Logical Architecture of Continuous Querying

Data Flow with CQs

CQ Events

CQ Event Filter

Region Type Restrictions for CQs

Implementing Continuous Querying

Continuous Query Implementation

Managing Continuous Querying

Using CQs from a RegionService Instance

States of a CQ

CQ Management Options

Managing CQs and Durable Clients Using gfsh

Retrieving an Initial Result Set of a CQ

Transactions

Adherence to ACID Promises
Atomicity
Consistency
Isolation
Durability

Code Examples
Transaction within an Application
Transaction within a Function

Design Considerations
Colocate Partitioned Regions
Region Operations Return References
First Operation with Mixed Region Types
Allowing Transactions to Work on Persistent Regions
Mixing Transactions with Queries and Indexes
Mixing Transactions with Eviction
Mixing Transactions with Expiration
Mixing Transactions with Non-transactional Operations
Changing the Handling of Dirty Reads

Function Execution

How Function Execution Works
Where Functions Are Executed
How Functions Are Executed
Highly Available Functions
Function Execution Scenarios

Executing a Function in Tanzu GemFire
Write the Function Code
Register the Function Automatically by Deploying a JAR
Register the Function Programmatically
Run the Function
Write a Custom Results Collector
Targeting Single Members of a Member Group or Entire Member Groups

Distributed Lock Service
Creating the Distributed Lock Service
Acquiring and Releasing Locks
Lock Grantor
Error Handling
Deadlock Handling
Lock Acquisition Best Practices

Developing REST Applications for Tanzu GemFire

GemFire REST API Overview

Prerequisites and Limitations for Writing REST Applications

Setup and Configuration
REST API Libraries
Enabling the REST API
Enabling the REST API on Multiple Servers
Starting the REST API Service
Configure PDX for your cluster

Start the REST API Service on One or More Servers
Verify That The Service is Running
Implementing Authentication

Using the Swagger UI to Browse REST APIs

Developing REST Applications

Working with Regions
Listing Available Regions
Reading Region Data
Adding or Modifying Region Data
Deleting Region Data
Working with Queries
Listing Queries
Creating a New Query
Executing a Prepared Query
Modifying a Prepared Query
Deleting a Prepared Query
Executing an Ad-Hoc Query
Working with Functions
Listing Functions
Executing Functions

Sample REST Applications

#1. REST Java Client (RestClientApp.java)
#1a. Tanzu GemFire Cache Java Client (MyJavaClient.java)
#1b. REST Client Utilities (RestClientUtils.java)
#1c. Date and Time Utilities (DateTimeUtils.java)
#1d. Person Class (Person.java)
#1e. Gender Class (Gender.java)
#2. Ruby REST Client (restClient.rb)
#3. Python REST Client (restClient.py)

Troubleshooting and FAQ

Checking if the REST API Service is Up and Running
Key Types and JSON Support
Unsupported JSON Example 1
Unsupported JSON Example 2
Unsupported JSON Example 3
Unsupported JSON Example 4

Tanzu GemFire REST API Reference

Region Endpoints

GET /gemfire-api/v1

Resource URL
Parameters
Example Request
Example Success Response
Error Codes

GET /gemfire-api/v1/{region}

Resource URL
Parameters
Example Request

Example Success Response
Error Codes

GET /gemfire-api/v1/{region}/keys
Resource URL
Parameters
Example Request
Example Success Response
Error Codes

GET /gemfire-api/v1/{region}/{key}
Resource URL
Parameters
Example Request
Example Responses
Error Codes

GET /gemfire-api/v1/{region}/{key1},{key2},…,{keyN}
Resource URL
Parameters
Example Requests
Example Success Responses
Error Codes
Example Error Response
Implementation Notes

HEAD /gemfire-api/v1/{region}
Resource URL
Parameters
Example Request
Example Success Response
Error Codes

POST /gemfire-api/v1/{region}?key=
Resource URL
Parameters
Example Request
Example Success Response
Error Codes
Example Error Response

PUT /gemfire-api/v1/{region}/{key}
Resource URL
Parameters
Example Request
Example Success Response
Error Codes
Implementation Notes

PUT /gemfire-api/v1/{region}/{key1},{key2},…{keyN}
Resource URL
Parameters
Example Request
Example Success Response
Error Codes

PUT /gemfire-api/v1/{region}/{key}?op=REPLACE

- Resource URL
- Parameters
- Example Request
- Example Success Response
- Error Codes

PUT /gemfire-api/v1/{region}/{key}?op=CAS

- Resource URL
- Parameters
- Example Request
- Example Success Response
- Error Codes
- Example Error Response
- Implementation Notes

DELETE /gemfire-api/v1/{region}

- Resource URL
- Parameters
- Example Request
- Example Success Response
- Error Codes

DELETE /gemfire-api/v1/{region}/{key}

- Resource URL
- Parameters
- Example Request
- Example Success Response
- Error Codes

DELETE /gemfire-api/v1/{region}/{key1},{key2},…{keyN}

- Resource URL
- Parameters
- Example Request
- Example Success Response
- Error Codes

Query Endpoints

GET /gemfire-api/v1/queries

- Resource URL
- Parameters
- Example Request
- Example Response
- Error Codes

POST /gemfire-api/v1/queries?id=&q=

- Resource URL
- Parameters
- Example Request
- Example Success Response
- Error Codes

POST /gemfire-api/v1/queries/{queryId}

- Resource URL
- Parameters

Example Request
Example Success Response
Error Codes

PUT /gemfire-api/v1/queries/{queryId}
Resource URL
Parameters
Example Request
Example Success Response
Error Codes
Implementation Notes

DELETE /gemfire-api/v1/queries/{queryId}
Resource URL
Parameters
Example Request
Example Success Response
Error Codes

GET /gemfire-api/v1/queries/adhoc?q=
Resource URL
Parameters
Example Request
Example Success Response
Error Codes

Function Endpoints

GET /gemfire-api/v1/functions
Resource URL
Parameters
Example Request
Example Success Response
Error Codes

POST /gemfire-api/v1/functions/{functionId}
Resource URL
Parameters
Example Requests
Example Success Responses
Error Codes

Administrative Endpoints

[HEAD | GET] /gemfire-api/v1/ping
Resource URL
Parameters
Example Request
Example Success Response
Error Codes

GET /gemfire-api/v1/servers
Resource URL
Parameters
Example Request
Example Success Response

Error Codes

Tools, Modules, and Extensions

gfsh (GemFire SHell)

What You Can Do with gfsh

Starting gfsh

Start gfsh with a Custom Wrapper Script

Configuring the gfsh Environment

Configuring gfsh Security

Configuring gfsh Environment Variables

Configuring gfsh Session Logging

Member Log Files

Viewing Standard Output and Standard Error

Tab Completion

Command History and gfsh.history

JMX Manager Update Rate and System Monitoring

Formatting of Results

Useful gfsh Shell Variables

Basic Shell Features and Command-Line Usage

Tutorial—Performing Common Tasks with gfsh

Quick Reference of gfsh Commands by Functional Area

Basic GemFire gfsh Commands by Functional Area

Configuration Commands

Data Commands

Deployment Commands

Disk Store Commands

Durable CQ and Client Commands

Function Execution Commands

Gateway (WAN) Commands

GemFire AsyncEventQueue Commands

GemFire Monitoring and Managing Commands

Index Commands

JMX Connection Commands

Locator Commands

PDX Commands

Region Commands

Server Commands

gfsh Command Help

alter

alter async-event-queue

alter disk-store

alter gateway-sender

alter query-service

alter region

alter runtime

backup disk-store

cancel

- **cancel query**
- **cancel wan-copy region**
  - Syntax
  - Parameter
  - Example

change loglevel

clear defined indexes

close

- **close durable-client**
- **close durable-cq**

compact

- **compact disk-store**
- **compact offline-disk-store**

configure

- **configure pdx**

connect

create

- **create async-event-queue**
- **create defined indexes**
- **create disk-store**
- **create gateway-receiver**
- **create gateway-sender**
- **create index**
- **create initializer**
- **create jndi-binding**
- **create region**

debug

define index

deploy

describe

- **describe async-event-queue**
- **describe client**
- **describe config**
- **describe connection**
- **describe disk-store**
- **describe gateway-receiver**
- **describe gateway-sender**
- **describe jndi-binding**
- **describe member**
- **describe offline-disk-store**
- **describe query**
- **describe query-service**
- **describe region**
- **describe wan-copy region**
  - Syntax

Parameters
Output
Examples 1
Example 2

destroy
destroy async-event-queue
destroy disk-store
destroy function
destroy gateway-receiver
destroy gateway-sender
destroy index
destroy initializer
destroy jndi-binding
destroy region

disconnect

echo

execute function
execute function

exit

export
export cluster-configuration
export config
export data
export logs
export offline-disk-store
export stack-traces

gc

get

help

hint

history

import
import cluster-configuration
import data

list
list async-event-queues
list clients
list deployed
list disk-stores
list durable-cqs
list functions
list gateways
list indexes
list initializers
list jndi-binding
list members

list queries
list regions
list wan-copy region
Syntax
Parameters
Example 1

load-balance gateway-sender

locate entry
locate entry

netstat

pause gateway-sender

pdx rename

put

query

rebalance

remove

restore redundancy

resume
resume async-event-queue-dispatcher
resume gateway-sender

revoke missing-disk-store

run

set variable

sh

show
show dead-locks
show log
show metrics
show missing-disk-stores
show statistics
show subscription-queue-size

shutdown

sleep

start
start gateway-receiver
start gateway-sender
start jconsole
start jvisualvm
start locator
start pulse
start server
Examples
start vsd

status
status cluster-config-service
status gateway-receiver
status gateway-sender
status locator
status redundancy
status server
stop
stop gateway-receiver
stop gateway-sender
stop locator
stop server
undeploy
upgrade
validate offline-disk-store
version
wan-copy region
Parameters
Example Commands
Example Output 1
Example Output 2
Example 3
Example 4
Error Messages
Region Expression Examples
Creating and Running gfsh Command Scripts
Running gfsh Scripts
Running gfsh Commands on the OS Command Line
Running Multiple gfsh Commands on the OS Command Line
Specifying JSON within Command-Line Options
Mapping cache.xml Elements to gfsh Configuration Commands
Gemcached
How Gemcached Works
Deploying and Configuring a Gemcached Server
Embedding a Gemcached server in a Tanzu GemFire Java Application
Starting a Gemcached Server Using a gfsh Command
Configuring a Gemcached Server with the gemfire.properties File
Advantages of Gemcached over Memcached
Pulse
Running Pulse in Embedded Mode (Quick Start)
Hosting Pulse on a Web Application Server
Configuring Pulse Authentication
Configuring Pulse to use HTTPS
Configuring Pulse to use Security Manager

Configuring Pulse to use a Custom Security Profile
Configuring Pulse to use an OAuth Authentication Provider

Using Pulse Views

Cluster View
Member View
Region View
Data Browser
Alerts Widget

Visual Statistics Display

VSD Overview

Installing and Running VSD

Install VSD
Configure Statistics Sampling in GemFire
Start VSD
Load a Statistics File into VSD
Maintain a Current View of the Data File
About Statistics
.gfs Time Zone Information for Matching Statistics to Log Files

Viewing Statistics in VSD

Statistic Levels
Select Statistics for Viewing
Using VSD Chart Templates
Chart Menu (Chart Window)
Line Menu (Chart Window)
Customizing Your VSD Chart
View Statistic Information

Quick Guide to Useful Statistics

Runtime Configuration
Resources
Throughput for Different Operations

Reference

GemFire Properties

Using Non-ASCII Strings in GemFire Property Files

cache.xml

cache.xml Quick Reference

Cache XML Requirements
Variables in cache.xml
Configuration Quick Reference

Element Hierarchy

Element Reference

cache Attributes
cache-transaction-manager
transaction-listener
transaction-writer
dynamic-region-factory
disk-dir

gateway-sender
gateway-event-filter
gateway-event-substitution-filter
gateway-receiver
gateway-conflict-resolver
async-event-queue
async-event-listener
cache-server
client-subscription
custom-load-probe
pool
locator
server
disk-store
disk-dirs
disk-dir
pdx
pdx-serializer
region-attributes
key-constraint
value-constraint
region-time-to-live
expiration-attributes
custom-expiry
region-idle-time
expiration-attributes
custom-expiry
entry-time-to-live
expiration-attributes
custom-expiry
entry-idle-time
expiration-attributes
custom-expiry
partition-attributes
partition-resolver
partition-listener
fixed-partition-attributes
membership-attributes
required-role
subscription-attributes
cache-loader
cache-writer
cache-listener
compressor
eviction-attributes
lru-entry-count
lru-heap-percentage
lru-memory-size
<jndi-bindings
jndi-binding
config-property

config-property-name
config-property-type
config-property-value
region
index
lucene:index
lucene:field
entry
key
<string>
declarable
value
<string>
declarable
region
function-service
function
resource-manager
serialization-registration
serializer
instantiator
backup
initializer
declarable
class-name and parameter
declarable
string

Element Hierarchy

Element Reference

client-cache Attributes
cache-transaction-manager
transaction-listener
transaction-writer
pool
locator
server
socket-factory
disk-store
disk-dirs
disk-dir
pdx
pdx-serializer
region-attributes
key-constraint
value-constraint
region-time-to-live
expiration-attributes
custom-expiry
region-idle-time
expiration-attributes

custom-expiry
entry-time-to-live
expiration-attributes
custom-expiry
entry-idle-time
expiration-attributes
custom-expiry
cache-loader
cache-writer
cache-listener
eviction-attributes
lru-entry-count
lru-heap-percentage
lru-memory-size
jndi-bindings
jndi-binding
config-property
config-property-name
config-property-type
config-property-value
region
region-attributes
index
entry
key
string
declarable
value
string
declarable
region
function-service
function
resource-manager
serialization-registration
serializer
instantiator
initializer

Region Shortcuts

Region Shortcuts Reference

LOCAL
- Description
- Default Attributes
- gfsh Command Example
- XML Definition of LOCAL

LOCAL_HEAP_LRU
- Description
- Default Attributes
- gfsh Command Example
- XML Definition of LOCAL_HEAP_LRU

LOCAL_OVERFLOW
Description
Default Attributes
gfsh Command Example
XML Definition of LOCAL_OVERFLOW
LOCAL_PERSISTENT
Description
Default Attributes
gfsh Command Example
XML Definition of LOCAL_PERSISTENT
LOCAL_PERSISTENT_OVERFLOW
Description
Default Attributes
gfsh Command Example
XML Definition of LOCAL_PERSISTENT_OVERFLOW
PARTITION
Description
Default Attributes
gfsh Command Example
XML Definition of PARTITION
PARTITION_HEAP_LRU
Description
Default Attributes
gfsh Command Example
XML Definition of PARTITION_HEAP_LRU
PARTITION_OVERFLOW
Description
Default Attributes
gfsh Command Example
XML Definition of PARTITION_OVERFLOW
PARTITION_PERSISTENT
Description
Default Attributes
gfsh Command Example
XML Definition of PARTITION_PERSISTENT
PARTITION_PERSISTENT_OVERFLOW
Description
Default Attributes
gfsh Command Example
XML Definition of PARTITION_PERSISTENT_OVERFLOW
PARTITION_PROXY
Description
Default Attributes
gfsh Command Example
XML Definition of PARTITION_PROXY
PARTITION_PROXY_REDUNDANT
Description
Default Attributes
gfsh Command Example
XML Definition of PARTITION_PROXY_REDUNDANT
PARTITION_REDUNDANT

Description
Default Attributes
gfsh Command Example
XML Definition of PARTITION_REDUNDANT
PARTITION_REDUNDANT_HEAP_LRU
Description
Default Attributes
gfsh Command Example
XML Definition of PARTITION_REDUNDANT_HEAP_LRU
PARTITION_REDUNDANT_OVERFLOW
Description
Default Attributes
gfsh Command Example
XML Definition of PARTITION_REDUNDANT_OVERFLOW
PARTITION_REDUNDANT_PERSISTENT
Description
Default Attributes
gfsh Command Example
XML Definition of PARTITION_REDUNDANT_PERSISTENT
PARTITION_REDUNDANT_PERSISTENT_OVERFLOW
Description
Default Attributes
gfsh Command Example
XML Definition of PARTITION_REDUNDANT_PERSISTENT_OVERFLOW
REPLICATE
Description
Default Attributes
gfsh Command Example
XML Definition of REPLICATE
REPLICATE_HEAP_LRU
Description
Default Attributes
gfsh Command Example
XML Definition of REPLICATE_HEAP_LRU
REPLICATE_OVERFLOW
Description
Default Attributes
gfsh Command Example
XML Definition of REPLICATE_OVERFLOW
REPLICATE_PERSISTENT
Description
Default Attributes
gfsh Command Example
XML Definition of REPLICATE_PERSISTENT
REPLICATE_PERSISTENT_OVERFLOW
Description
Default Attributes
gfsh Command Example
XML Definition of REPLICATE_PERSISTENT_OVERFLOW
REPLICATE_PROXY
Description

Default Attributes
gfsh Command Example
XML Definition of REPLICATE_PROXY

Region Shortcuts Reference Table

Exceptions and System Failures

Memory Requirements for Cached Data

Core Guidelines for Tanzu GemFire Data Region Design

Memory Usage Overview
Calculating Application Object Overhead
Using Key Storage Optimization
Measuring Cache Overhead
Estimating Management and Monitoring Overhead
Determining Object Serialization Overhead
Calculating Socket Memory Requirements

GemFire Statistics List

Cache Performance (CachePerfStats)
Cache Server (CacheServerStats)
Client-Side Notifications (CacheClientUpdaterStats)
Client-to-Server Messaging Performance (ClientStats and ClientSendStats)
Client Connection Pool (PoolStats)
Continuous Querying (CqServiceStats)
Disk Space Usage (DiskDirStatistics)
Disk Store Statistics (DiskStoreStatistics)
Disk Usage and Performance (DiskRegionStatistics)
Distributed System Messaging (DistributionStats)
Distribution Statistics Related to Slow Receivers
Distributed Lock Services (DLockStats)
Function Execution (FunctionStatistics)
Gateway Queue (GatewaySenderStatistics)
Indexes (IndexStats)
Query-Independent Statistics on Indexes
Query-Dependent Statistics on Indexes
JVM Performance
Tanzu GemFire JVM Resource Manager (ResourceManagerStats)
JVM Java Runtime (VMStats)
JVM Garbage Collection (VMGCStats)
JVM Garbage Collector Memory Pools (VMMemoryPoolStats)
JVM Heap Memory Usage (VMMemoryUsageStats)
JVM Thread Statistics (VMThreadStats)
Locator (LocatorStats)
Lucene Indexes (LuceneIndexStats)
Off-Heap (OffHeapMemoryStats)
Operating System Statistics - Linux
Linux Process Performance (ProcessStats)
Linux Operating System (OperatingSystemStats)
System Processors (ProcessorStats)
Linux Networking Statistics (NetworkInterfaceStats)
Disk / Filesystem Statistics (DiskStatistics)

Partitioned Regions (PartitionedRegion<partitioned_region_name>Statistics)
Partitioned Region Statistics on Partition Messages
Partitioned Region Statistics on Data Entry Caching
Partitioned Region Statistics on Redundancy
Region Entry Eviction – Count-Based (LRUStatistics)
Region Entry Eviction – Heap-based eviction (HeapLRUStatistics)
Region Entry Eviction – Size-based (MemLRUStatistics)
Server Notifications for All Clients (CacheClientNotifierStatistics)
Server Notifications for Single Client (CacheClientProxyStatistics)
Server-to-Client Messaging Performance (ClientSubscriptionStats)
Statistics Collection (StatSampler)
Transaction Reference Material
JTA Global Transactions with Tanzu GemFire
Coordinating with External JTA Transaction Managers
How to Run a JTA Transaction Coordinated by an External Transaction Manager
Using Tanzu GemFire as the “Last Resource” in a Container-Managed JTA Transaction
How to Run JTA Transactions with Tanzu GemFire as a “Last Resource”
Behavior of GemFire Cache Writers and Loaders Under JTA
Turning Off JTA Transactions
Experimental Features
Cluster Management Service
Cluster Management Configuration
Authentication and Authorization
Java API
Golang Command Line Interface
Automated Rebalance
Glossary
ACK wait threshold
administrative event
API
application program
attribute
attribute path
blocking
cache
cache-local
cache.xml
cache event
cache listener
cache loader
cache miss
cache server
cache transaction
cache writer
client

client region
cluster configuration service
collection
commit
concurrency-level
conflation
connection
consumer
coordinator
data accessor
data entry
data fabric
data-policy
data region (region)
data store
deadlock
destroy
disk region
disk-store
distributed cache
distributed system
distributed-ack scope
distributed-no-ack scope (Deprecated - no replacement)
entry
entry key
entry value
event
eviction-attributes
expiration
expiration action
factory method
forced disconnect
gateway receiver
gateway sender
gemfire.properties
global scope
global transaction
HTTP
idle timeout
initial capacity
invalid
invalidate
JDBC
JMX
JNDI
JTA
JVM
key constraint
listener
load factor
local

local scope
locator
LRU
machine
member
message queue
mirroring
named region attributes
netLoad
netSearch
netWrite
network partitioning
OQL
off-heap memory
overflow
oplog / operation log
partition
partitioned region
peer
persistent region
persistent-partition
persistent-replicate
producer
pull model
push model
query string
race condition
range-index
region
region attributes
region data
region entry
region shortcut
remote
replicated region
replicate
resource manager
rollback
scope
SELECT statement
serialization
server
server group
server connection pool
socket
SQL
SSL
standalone distributed system
statistics enabled
struct
structure-index

system member
TCP
timeout
time-to-live
transaction
transaction listener
transaction writer
transactional view
transport layer
TTL
URI
user attribute
value constraint
value-index
view
Virtual Machine
VMware virtual machine
XML
XML schema definition
XPath
XSD

Documentation Legal Notice

Documentation

This topic outlines the VMware Tanzu GemFire documentation set.

This documentation set describes product concepts and provides complete installation and configuration instructions for Tanzu GemFire.

Tanzu GemFire is a data management platform that provides real-time, consistent access to data-intensive applications throughout widely distributed cloud architectures.

Tanzu GemFire 10.2

- Tanzu GemFire 10.2 Release Notes
- Supported Configurations and System Requirements
- Getting Started
    - Installing Tanzu GemFire
    - Upgrading Tanzu GemFire
    - Tanzu GemFire in 15 Minutes or Less
- Configuring and Running a Cluster
- Basic Configuration and Programming
- Topologies and Communication
- Managing Tanzu GemFire
- Developing with Tanzu GemFire
- Developing REST Applications for Tanzu GemFire
- Tools, Modules, and Extensions
- Reference
- Experimental Features

API Reference Documentation

- VMware Tanzu GemFire Java API
- VMware Tanzu GemFire Developer REST API

Documentation of Related Products

- VMware Tanzu Gemfire on Kubernetes
- VMware Tanzu GemFire on Tanzu Platform
- VMware Tanzu GemFire Management Console
- VMware Tanzu GemFire Search
- VMware Tanzu GemFire Vector Database

- VMware Tanzu GemFire Native Client
- VMware Tanzu GemFire Session Management
- VMware Spring Data for Tanzu GemFire
- VMware Spring Boot for Tanzu GemFire
- VMware Spring Session for Tanzu GemFire
- VMware Spring Integration for Tanzu GemFire
- VMware Spring Cloud Data Flow for Tanzu GemFire
- Deploy and Manage VMware Tanzu GemFire on vSphere

Release Notes

This topic contains the release notes for VMware Tanzu GemFire.

Upgrade Restrictions

I want to upgrade to:	If my current version is:	What can I upgrade to?
10.2	9.x	Must upgrade 9.x -> 10.1 -> 10.2
10.2	10.x	No restrictions. You can upgrade directly to 10.2.

New
Default Setting Changes
Deprecations
Known Issues
Resolved Issues
Security Fixes

New in Tanzu GemFire 10.2

Enhanced WAN Copy Efficiency with Wildcard Region Selection

The wan-copy region command now supports wildcards and regular expressions, allowing you to initiate WAN copies for multiple matching regions in a single step. This simplifies the WAN copy process for large and complex deployments. Three new commands are available:

- cancel wan-copy region
- list wan-copy region
- describe wan-copy region

For more information, see wan-copy region.

WAN Copy Improvements

The wan-copy region command now ensures that any missing PDX types are sent to the remote site before copying region data.

The wan-copy region command now honors GatewayEventFilter and GatewayEventSubstitutionFilter configurations on the gateway sender, using substitute values and filtering events accordingly.

There are new statistics in GatewaySenderStats for WAN copies:

- failedWanCopyRegionTime (nanoseconds)
- regionsCopied (operations)
- regionsFailedToBeCopied (operations)
- wanCopyRegionFilterEventsTime (nanoseconds)

- wanCopyRegionInProgress (operations)
- wanCopyRegionTime (nanoseconds)

Message Loss Detection and New Stats

GemFire now attempts to recover from situations where network middleware disconnects TCP connections between GemFire peers. In most cases, GemFire will reconnect and retry messages lost due to TCP disconnects. In some rare cases, after reconnecting, GemFire may detect that messages sent to that member were lost. In that case, an error message will indicate the name of the lost message.

New statistics include: messageLossEvents, messagesLost. For more information, see *Message Loss Detected* in the Log Messages and Solutions and the Statistics List.

GemFire messaging now uses connection pooling

GemFire's peer-to-peer messaging now pools connections when conserve-sockets is false, which is the default. This helps reduce the number of sockets and threads used for server-to-server communication across the GemFire cluster.

Enhanced Failure Handling and Retry Support for WAN Replication

The REMOVE_FROM_QUEUE_ON_EXCEPTION property is deprecated and replaced by the GatewayEventFailureListener callback interface. Enhancements include:

- Custom handling of failed WAN events.
- Support for storing failed events in a user-defined region.
- New gateway receiver properties for retry attempts and intervals.
- Improved exception handling logic to distinguish between recoverable and terminal failures.

For more information, see Handling WAN Replication Errors.

Improved Redaction

Redaction of sensitive fields in logs and gfsh has been improved.

OS-Level Statistics Migrated to OSHI

OS-level statistics now use the OSHI library, replacing LinuxSystemStats and LinuxProcessStats. Benefits include:

- Better Windows support
- More detailed hardware, networking, processor, and disk stats
- More consistent results across platforms

Cluster Health Documentation

New documentation has been added that describes six key health factors in Tanzu GemFire that help assess system stability:

- JVM Pauses High
- CPU Steal High
- Resource Manager Health Factors
- Reply Timeouts High

- Partitioned Region Redundancy Low
- Disk Low

Tanzu GemFire on vSphere reference architecture documentation

The *Deploy and Manage VMware Tanzu GemFire on vSphere* reference architecture documentation is now available here.

Off platform WAN access

Off platform access extends GemFire's SNI proxy support to WAN connections between clusters.

- Remote locators may be defined with a sni://... URI in order to discover and connect to WAN sites that are behind a SNI-capable proxy.

Improved TLS 1.3 handshake performance

The performance of TLS 1.3 handshaking has been improved in situations where many peer-to-peer connections are being created.

Improved logging when a member is forcibly disconnected

The ForcedDisconnectException has been enhanced to provide more information to assist in determining the cause of the forced disconnection. The log message will be similar to:

```
cluster membership failed due to
org.apache.geode.ForcedDisconnectException: Exiting due to possible network partition event due to loss of 1 cache processes:
[192.168.37.245(server1:190)<v1>:50571]. No thread delays observed on this member. System metrics in the last 59 seconds
{CPU usage=33.1%, Load Average=30.18, Membership thread delay=0 seconds, TCP segment retransmits=0%, GC collection ti
me=0 ms}
    at gemfire//org.apache.geode.distributed.internal.ClusterDistributionManager$DMListener.membershipFailure(ClusterDistrib
utionManager.java:2156)
Caused by: org.apache.geode.distributed.internal.membership.api.MemberDisconnectedException: Exiting due to possible netw
ork partition event due to loss of 1 cache processes: [192.168.37.245(server1:190)<v1>:50571]
    at gemfire//org.apache.geode.distributed.internal.membership.gms.GMSMembership$ManagerImpl.forceDisconnect(GMS
Membership.java:1774)
    ...
```

New statistics

The following new statistics have been added:

- DistributionStats:
  - membershipClientThreadDelayDuration (milliseconds)
  - membershipClientThreadPauses (threadPauses)
  - messageChunksSentSharedOrdered (fragments)
  - messageChunksSentSharedUnordered (fragments)
  - messageChunksSentUnshared (fragments)
  - messageLossEvents (events)
  - messageSendDelays (delays)
  - messagesLost (messages)
  - messagesResent (messages)

    - p2pSocketTimeouts (timeouts)
    - receiverCloses (sockets)
    - receiverCreates (sockets)
    - receiverSocketConnects (sockets)
    - receiversShared (sockets/threads)
    - receiversUnshared (sockets/threads)
    - reconnectAttempts (reconnects)
    - reconnects (reconnects)
    - senderAsyncCreateRequests (operations)
    - senderCloses (sockets)
    - senderCreates (sockets)
    - senderCreatesInProgress (operations)
    - senderSocketConnects (sockets)
    - sendersPurgedFromPool (operations)
    - sendersUnshared (sockets)
    - sendersUnsharedDisconnectChecks (operations)
    - sendersUnsharedThreadOwned (sockets)
    - sentDirectReplyMessages (messages)
    - sentMessageQueueSize (messages)
    - sentMessagesAcked (messages)
    - sentSharedOrderedMessages (messages)
    - sentSharedOrderedMessagesInProgress (threads)
    - sentSharedOrderedMessagesTime (nanoseconds)
    - sentUnsharedMessages (messages)
    - sentUnsharedMessagesInProgress (threads)
    - sentUnsharedMessagesTime (nanoseconds)
    - sentUnsharedThreadOwnedMessages (messages)
- GatewaySenderStats:
    - failedWanCopyRegionTime (nanoseconds)
    - regionsCopied (operations)
    - regionsFailedToBeCopied (operations)
    - wanCopyRegionFilterEventsTime (nanoseconds)
    - wanCopyRegionInProgress (operations)
    - wanCopyRegionTime (nanoseconds)
- CachePerfStats:
    - expiryDelayTime (milliseconds)
    - expiryDelays (operations)

    - expiryTasks (entries)
    - expiryThreads (threads)
    - stateFlushOperationTime (nanoseconds)
- ResourceManagerStats:
    - entryExpirationTime (nanoseconds)
    - entryExpirations (operations)
    - entryExpirationsInProgress (operations)
    - entryExpireDestroys (operations)
    - entryExpireInvalidates (operations)
    - entryExpireLocalDestroys (operations)
    - entryExpireLocalInvalidates (operations)
    - latestLastAccessTimeChecks (operations)
- CacheServerStats:
    - processRegisterInterestRequestTime (nanoseconds)
    - readRegisterInterestRequestTime (nanoseconds)
    - writeRegisterInterestResponseTime (nanoseconds)
- OperatingSystemStats
- ProcessStats
- ProcessorStats
- DiskStatistics
- NetworkInterfaceStats

HTTP authentication

All authenticated HTTP requests use standard headers (including basic authentication and token-based authentication).

Create and delete initializers using gfsh

In addition to cache.xml, initializers can be managed directly with gfsh.

- Use gfsh create initializer --initializer=CLASS_NAME to create an initializer
- Additional gfsh commands are destroy initializer and list initializers

Increase the maximum socket lease time to 7 days

The maximum socket lease time is increased to 7 days.

ResourceManager updates and support for Generational ZGC

The ResourceManager has been updated to use collectionUsage instead of currentUsage for ZGC since collectionUsage is a better approximation of long lived heap size.

The gfsh start locator and start server commands enable ZGC by default on JDK 17–20, and GenZGC on JDK 21 and later.

Improvements to gfsh deadlock detection Command

The gfsh deadlock detection command now:

- Records how long each thread has been blocked during deadlock analysis and displays the top five longest-waiting threads in the output.
- Collects stack traces only when a deadlock is detected, improving performance and reducing overhead during the detection phase.

Accounting for JVM pauses has been improved

StatSampler now records delays caused by CPU starvation as a jvmPause.

Consistent path naming in GemFire tgz bundles

Removed the superfluous ./ from paths in all GemFire-related tgz bundles. This allows for consistent use of tar -strip-components=X to get to the same level as bundles without a ./ in the path.

Default Setting Changes

The following configuration defaults have been updated in this release:

- The default value for socket lease time is updated to 1 minute.
- The property gemfire.ON_DISCONNECT_CLEAR_PDXTYPEIDS now defaults to true, causing the client-side PDX registry to automatically refresh when connecting to a server.
- Prometheus metrics are now exposed through the member's configured http-service-port. The Prometheus metrics endpoint is enabled by default for members with the HTTP service enabled (for example, when starting the REST API). For more information, see Prometheus Metrics Endpoint.
- Stats collection for servers is now enabled by default with a default sample rate of 1000 milliseconds. The default location for the stats archive file is {workingDir}/stats/{serverName}.gfs. Override this location by setting the statistic-archive-file property.

Deprecations

The following features, properties, or APIs are deprecated and will be removed in a future release.

- The spring-xml-location property in gfsh start server is marked for future removal; no replacement is planned.
- Using -1, 0, or 255 for members that host a gateway sender is deprecated. A distributed-system-id of -1 (the default) indicates that WAN replication is disabled. To enable WAN replication, set the member's distributed-system-id to a value from 1 to 254 (inclusive).
- Using 0 or 255 as the distributed-system-id is deprecated. If a locator or server is started using any method (including gfsh) with the distributed-system-id set to 0 or 255, one of the following warnings is displayed:

  ```
  Warning: Using 0 as the distributed-system-id is deprecated as of GemFire 10.2

  Warning: Using 255 as the distributed-system-id is deprecated as of GemFire 10.2
  ```

- The REMOVE_FROM_QUEUE_ON_EXCEPTION property is deprecated.

- Adding fixed custom partitioning for partitioned regions is deprecated and will be removed in upcoming releases.
- Fixed custom partitioning is deprecated and will be removed in upcoming releases.
- Gemcached is deprecated and will be removed in upcoming releases.
- The ability to have transaction events delivered atomically (i.e. within a single batch) is deprecated. This includes deprecating the following APIs and properties:
    - InternalRegion.invokeTXCallbacks
    - GatewaySenderFactory.setGroupTransactionEvents
    - GatewaySenderAttributes.setGroupTransactionEvents
    - GatewaySenderAttributes.setRetriesToGetTransactionEventsFromQueue
    - GatewaySenderAttributes.mustGroupTransactionEvents
    - GatewaySenderAttributes.getRetriesToGetTransactionEventsFromQueue
    - group-transaction-events attribute on gateway-sender cache xml element
    - gemfire.get-transaction-events-from-queue-retries system property
    - gemfire.get-transaction-events-from-queue-wait-time-ms system property
- The DISTRIBUTED_NO_ACK region scope has been deprecated. There is no replacement.
- The statistic-sampling-enabled property and the alter runtime --enable-statistics parameter are deprecated as statistic collection will be enabled by default. It will be removed in upcoming releases.
- Region-wide expiration is deprecated. There is no replacement. This includes deprecating the following:
    - region-time-to-live cache xml element
    - region-idle-time cache xml element
    - create region --region-time-to-live-expiration parameter
    - create region --region-time-to-live-expiration-action parameter
    - create region --region-idle-time-expiration parameter
    - create region --region-idle-time-expiration-action parameter
    - alter region --region-time-to-live-expiration parameter
    - alter region --region-time-to-live-expiration-action parameter
    - alter region --region-idle-time-expiration parameter
    - alter region --region-idle-time-expiration-action parameter

---

Known Issues

- GEM-15797: A region data export may hang when a cluster member crashes while exporting. The workaround is to restart the member performing the export.
- GEM-15512: In the rare case when a server is kicked out of the cluster (force-disconnect) and then it is kicked out again while it is trying to reconnect (auto-reconnect), it can keep running but be unavailable for cache operations. The workaround is to restart the server.

- GEM-15671: When a primary gateway sender fails while distributing a batch destroy operation to its secondary gateway senders, certain issues may arise.

If a secondary sender restarts and obtains its initial image from another secondary that has just become the new primary, the result can vary depending on the timing of the batch destroy and the queue size of the new primary.

In some cases, events that were already processed might remain in the restarted secondary's SerialGatewaySender queue instead of being removed.

If that secondary later becomes the primary, these previously processed events might be treated as possible duplicates and re-delivered.

- GEM-15927: In the WAN gateway receiver site, when a member hosting a replicate region restarts and receives a retry from the remote SerialGatewaySender due to a failover, in some cases, this can lead to data inconsistency among the receiver-side members.

Resolved Issues

- GEM-5473: Resolved an issue where two members could have the same lockGrantorVersion by keeping track of the last known lock grantor version.
- GEM-6648: Resolved an issue where exceptions in Client were not handled correctly during shutdown.
- GEM-6662: Resolved an issue where client functions that fail with an exception perform expensive locator requests when calculating retry attempts.
- GEM-6710: Statistics jvmPause detection now also accounts for the time taken to collect a sample.
- GEM-6847: Resolved an issue where a NullPointerException was thrown during a self-join when a region had a Compact or Range index.
- GEM-6884: Added ClassLoader Modules to describe member --verbose command.
- GEM-6893: Resolved an issue where the cache was not closed when file descriptors were exhausted.
- GEM-6895: Resolved an issue where the alert state was not updated when the file descriptor threshold was reached. Alerts are now logged and visible using the gfsh describe member command.
- GEM-6922: Logs and exceptions from the shiro package are now routed to the Tanzu GemFire security log.
- GEM-6981: Reduced lock contention and Integer allocations in OpLog.
- GEM-6982: Reduced frequent transient Integer allocations in ThreadLocal<Integer> by replacing it with ThreadLocalInt.
- GEM-6998: Improved the Exception message thrown when gateways encounter a PDX type conflict.
- GEM-7008: Reduced transient object allocations in PersistentOplogSet.
- GEM-7011: Updated JVM pause time detection to include stat sampling duration; describe member now displays pause alerts only if they occurred in the last 10 minutes.
- GEM-7016: Reduced transient object allocations in DiskStoreImpl.
- GEM-7021: Cleaned up the use of Assert in various DLock-related classes to avoid unnecessary allocations.

- GEM-7045: Deprecated the use of DSIDs 0 and 255.
- GEM-7046: Deprecated using -1 as the gateway sender DSID, even when PDX is not in use.
- GEM-7106: Renamed CacheServerStats.connectionThreads to clientRequestsInProgress. connectionThreads now refers to the number of threads available to process client requests.
- GEM-7112: Resolved an issue that caused the gfsh list regions command to not show region status correctly when the region had not fully initialized on all members.
- GEM-7247: Resolved an issue where a MembershipClosedException could be thrown if a member was shutting down while initiating suspect member checks on another member.
- GEM-7256: Ensured that async event listeners honor the conserve-sockets setting.
- GEM-7306: Resolved an issue where a member could hang while waiting for bucket recovery during server restart when also waiting to sync with a killed member.
- GEM-7329: The gfsh create gateway-receiver command now checks that all members are running the same version.
- GEM-7370: Statistics.setSupplier methods initialize the statistic correctly from the Supplier's initial value.
- GEM-7375: Implemented SerialGatewaySenderEventProcessor.enqueueEvent to handle the processing of GatewaySenderQueueEntrySynchronizationOperations results.
- GEM-7376: Resolved an issue where stopping and starting durable CQ clients can incorrectly throw an IllegalStateException on the server.
- GEM-7377: Resolved an issue where registering a CQ, while the cache is also in process of closing, may throw an IllegalStateException instead of a CacheClosedException.
- GEM-7378: Resolved an issue that caused the client to stop searching for a primary interest queue prematurely.
- GEM-7389: Resolved an issue by caching ThreadIdentifier in EntryEventImpl to avoid duplicate creation.
- GEM-7396: Resolved an issue where a serial gateway sender could produce an out of memory error during system shutdown.
- GEM-7397: Resolved an issue where DistributionAdvisor incorrectly logs an assertion error during system shutdown.
- GEM-7405: Resolved an issue where client interest callbacks are not fired during client queue failover from one server to another.
- GEM-7440: Resolved an issue where tasks may be submitted to cancelled Timers during system shutdown.
- GEM-7442: Resolved an issue where ConcurrentModificationException can be thrown from AcceptorImpl during cache shutdown.
- GEM-7487: Resolved an issue where a failed RemotePutMessage can cause a PersistentReplicatesOfflineException to be thrown when no persistent members are offline.
- GEM-7521: Resolved an issue where alerting threads may give up on sending AlertListenerMessage too easily.
- GEM-7527: Resolved issues where interaction with the SystemTimer may produce exceptions such as IllegalStateException: This timer has been cancelled
- GEM-7547: Resolved issues when creating gateway senders using gfsh:

    - Do not allow illegal characters in the name of gateway senders.
    - Do not allow incompatible senders to be created in different members.
- GEM-7560: Resolved an issue where closing a CacheClientProxy may produce an exception like: IllegalStateException: Task already scheduled or cancelled.
- GEM-7567: Resolved an issue where DistributionStats reply timeout stat is not incremented when direct reply messages exceed the timeout.
- GEM-7575: Resolved an issue where gfsh shell does not auto connect to the locator if the SSL truststore and the keystore passwords are provided as –J= arguments.
- GEM-7588: Resolved an issue where adding a root cause (Exception) could produce a cause chain cycle.
- GEM-7609: Resolved an issue where FederatingManager did not handle RegionDestroyedException.
- GEM-7614: Resolved an issue where ThrowableUtils methods will hang if given a Throwable with a cause chain cycle.
- GEM-7621: Remove overly verbose warnings for deprecated disk region attributes triggered by internal use.
- GEM-7629: Resolved an issue where a destroy initiated from a client might incorrectly be retried and cause the originating client to have wrong data.
- GEM-7631: Resolved an issue where the gfsh connect command failed silently if the JMX manager was not started.
- GEM-7637: Resolved an issue where negative values for numeric fields were allowed when creating WAN gateway senders.
- GEM-7641: Resolved an issue where a member status was inconsistently reported in the gfsh list members and status locator/server commands. Status values can now be INITIALIZING, READY, or NOT AVAILABLE.
- GEM-7651: Resolved an issue where the REST API status command did not produce the correct error.
- GEM-7668: Improved retrieval of MBean attributes via the REST API. Use the Accept: text/plain header in REST calls to receive the response as plain text instead of a binary blob.
- GEM-7733: Resolved an issue where deserialization during low memory situations could produce errors with the message: Unknown header byte 0 during deserialization ....
- GEM-7744: Resolved an issue where closing the cache could cause additional memory allocations in TXFarSideCMTracker.
- GEM-7763: Converted unchecked BufferUnderflowException, in ByteBufferInputStream to throw IOException as appropriate, according to OutputStream method contracts.
- GEM-7764: Resolved an issue where members reported an incorrect status during auto-reconnect.
- GEM-7887: Validated that socket-buffer-size is not zero in the create gateway-sender and create gateway-receiver commands.
- GEM-7899: Resolved an issue where a PdxInstance could be shared between different lists even when copy-on-read is true.

- GEM-8023: Resolved an issue where ReflectionBasedAutoSerializer only allowed a comma and a space to separate class patterns.
- GEM-13414: Resolved an issue where Queries being sent to replicate-proxy regions return empty.
- GEM-13481: Suppressed ClosedSelectorException thrown from AcceptorImpl if the member is shutting down.
- GEM-13963: Resolved an issue where a member shutting down during GII could cause mismatched data.
- GEM-14025: Resolved an issue where removing an index before it was finished creating could cause errors.
- GEM-14109: Resolved a race condition that could prevent a system from reconnecting after being disconnected.
- GEM-14201: Resolved an issue that caused DiskStoreImpl.close(), PersistentOplogSet.close() and CacheServerImpl.stop() to ignore runtime exceptions due to a logic bug.
- GEM-14393: Fixed NullPointerException that could occur when a region was created as part of a CacheService initialization.
- GEM-14449: Resolved an issue where configuring sockets on Windows could throw an UnsupportedOperationException.
- GEM-14498: Resolved an issue where servers could hang during restart while concurrent index create/delete operations were in progress.
- GEM-14484: Removed JGroups. The JGroups library that enabled UDP communication in previous versions has been removed. Direct upgrades from Tanzu GemFire versions earlier than v10.0.0 to v10.2.0 are not supported, as those versions relied on UDP-based communication. For more information, see Upgrading from 9.x to Version 10.2.
- GEM-14508: Resolved an issue that caused the gfsh list members --verbose command to fail with the error: “String index out of range”.
- GEM-14513: Removed incorrect deprecation of min and max connections in GemFire cache xsd.
- GEM-14608: Improved expiration of events in HARegionQueue to avoid a possible out of memory situation.
- GEM-14643: Resolved an issue where the gfsh alter gateway-sender command could throw a NullPointerException.
- GEM-14648: Resolved an issue where the gfsh list regions command could incorrectly report a region’s data policy.
- GEM-15241: P2P sender now validates the receiver’s member ID during handshake to avoid possible incorrect message loss detection.
- GEM-15284: Resolved an issue where auto-reconnect might not work after a complete network outage and recovery.
- GEM-15288: Resolved an issue where a NullPointerException could be thrown from ManagementFunction during shutdown.
- GEM-15296: Resolved an issue where CacheClientProxyStats.messageQueueSize was not updated correctly.
- GEM-15303: Resolved an issue where the initial image request could throw InternalGemFireException when retrieving filter information.

- GEM-15373: Ensured that dispatched keys are sent to the secondary WAN queue before shutting down.
- GEM-15595: Resolved an issue where members suspected before the SuspectMembersMessage arrived were not included in the final check.
- GEM-15711: Resolved an issue where setting diskSynchronous was inconsistent between using cache.xml and creating a region with gfsh.
- GEM-15712: Prevented the cache from being created when use-cluster-configuration is true, but no cluster configuration can be retrieved.
- GEM-15857: Changed CacheClientNotifier.makePrimary to throw CacheClosedException instead of InternalGemFireError.
- GEM-15861: Improved performance by reducing the number of calls to PeerSenderImpl.accessed.
- GEM-9587: The socket-buffer-size option must now be greater than 0 in create gateway sender and create gateway receiver commands.
- GEM-14521: Resolved a race condition and deadlock issue that could occur when a region is created by a user or non-startup thread. The issue was fixed by ensuring the same cache lock is used for both the internal PDX metadata region and disk store.
- GEM-14724: Resolved a performance issue in the ThreadMonitor, where serial thread info collection, with lock-related data caused significant pauses and potential cluster instability. The following improvements were implemented:
    - Switched to batch-based thread info retrieval
    - Disabled collection of lock-related data
- GEM-14891: Added a back-off protocol for ThreadMonitor calls that exceed a threshold. This helps prevent the call from further impacting the system when under load.
- GEM-15458: Resolved an issue that caused a deadlock where bucket assignment was stalled because servers were waiting on the primary bucket election from each other during recovery.
- GEM-15976: Resolved an issue where GemFire servers could hang during cluster recovery from disk if multiple serial, persistent asynchronous event queues were defined and the servers were not shut down gracefully. When this issue occurs, each server might report that it is creating a different AsyncEventQueue and is waiting for newer data from other servers.

Security fixes

This topic lists all of the Common Vulnerabilities and Exposures (CVE) fixed in VMware Tanzu GemFire v10.2 and later.

> To see the full list of security advisories for VMware Tanzu products, go to the Security Advisories - Tanzu page on the Broadcom Support portal. To sign up to receive email notifications about security advisories and other alerts, follow the instructions in Subscribing to Broadcom product notifications and alerts.

v10.2.0 Security Fixes

Component	Upgraded Version	Resolved CVEs
Netty	4.1.127	CVE-2024-29025, CVE-2024-47535, CVE-2025-24970, CVE-2025-25193, CVE-2025-55163, CVE-2025-58056, CVE-2025-58057
Jetty	9.4.58.v20250814	CVE-2024-8184, CVE-2024-6763, CVE-2025-5115
Spring Framework	5.3.45	CVE-2024-22243, CVE-2024-22259, CVE-2024-22262, CVE-2024-38809, CVE-2024-38816, CVE-2024-38819, CVE-2024-38820, CVE-2025-22233
Spring Security	5.8.21	CVE-2024-22257
springdoc-openapi and Swagger UI	springdoc-openapi 1.8.0, Swagger UI 5.20.0	CVE-2024-45801, CVE-2024-47875, CVE-2025-26791
Apache Commons BeanUtils and Apache Commons Logging	BeanUtils 1.11.0, Logging 1.3.5	CVE-2025-48734
Nimbus JOSE+JWT	Updated to latest Connect2id release	CVE-2023-52428

Supported Configurations and System Requirements

This topic links to documentation that explains VMware Tanzu GemFire supported configurations and system requirements.

- Tanzu GemFire Supported Platforms

  Tanzu GemFire is supported on a variety of platforms.

- Host Machine Requirements

  Each machine must meet a set of system requirements.

- Java Support

  Tanzu GemFire requires a supported JRE or JDK on each system running Tanzu GemFire.

- Supported Platforms and System Requirements for Tools

  Tools, including Pulse and VSD, are supported on a variety of platforms.

> GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

Tanzu GemFire Supported Platforms

This topic lists the platforms where you can run VMware Tanzu GemFire.

You can run Tanzu GemFire production systems on the following platforms:

- Current Linux operating systems with recent kernel versions (4.18+).
- Current Windows Server versions, including 2012 R2, 2016, 2019, and 2022

You can run VMware Tanzu GemFire production systems on these platforms:

- Amazon Web Services (AWS)
- Microsoft Azure
- Google Cloud Platform
- Elastic Application Runtime
- Kubernetes

You can run VMware Tanzu GemFire development systems on these platforms:

- macOS
- Windows 10, Windows 11

> Use a consistent platform on all machines in your Tanzu GemFire cluster.

Host Machine Requirements

This topic lists the requirement for host machine to run VMware Tanzu GemFire. Each machine that will run Tanzu GemFire must meet the following requirements:

- Use the latest supported version of the guest operating system, for more information, see VMware Tanzu GemFire Supported Platforms.
- Use Java large paging in the guest OS. Configure Java on the guest OS to use large pages. Add the following command line option when launching Java:

  ```
  -XX:+UseLargePages
  ```

- Java SE Development Kit 8 with update 361 or a more recent version 8 update. The same versions are supported with OpenJDK. Tanzu GemFire is compatible with JDK 8, JDK 11, JDK 17, and JDK 21. See Java Support for details.
- An adequate per-user limit on the number of file descriptors; for Unix/Linux, the recommended soft limit is 8192, and the hard limit is 81920.
- An adequate per-user limit on the number of processes (nproc); for Unix/Linux, the recommended soft limit is 501408, with an unlimited hard limit.
- TCP/IP.
- A system clock set to the correct time and a time synchronization service such as Network Time Protocol (NTP). Correct time stamps permit the following activities:
  - Logs that are useful for troubleshooting. Synchronized time stamps ensure that log messages from different hosts can be merged to reproduce an accurate chronological history of a distributed run.
  - Aggregate product-level and application-level time statistics.
  - Accurate monitoring of the Tanzu GemFire system with scripts and other tools that read the system statistics and log files.
- The host name and host files are properly configured for the machine. The host name and host file configuration can affect gfsh and Pulse functionality.

  > Tanzu GemFire v10.1 deprecates Pulse in favor of VMware Tanzu GemFire management console. Pulse will be removed in a future release.

- Deactivate TCP SYN cookies. Most default Linux installations use SYN cookies to protect the system against malicious attacks that flood TCP SYN packets, but this feature is not compatible with stable and busy Tanzu GemFire clusters. Security implementations should instead seek to prevent attacks by placing Tanzu GemFire server clusters behind advanced firewall protection.

  To deactivate SYN cookies permanently:

  1. Edit the /etc/sysctl.conf file to include the following line:

     ```
     net.ipv4.tcp_syncookies = 0
     ```

     Setting this value to zero deactivates SYN cookies.

  2. Reload sysctl.conf:

```
sysctl -p
```

Java Support

VMware Tanzu GemFire v10 is certified for use with JDK 8, JDK 11, JDK 17, and JDK 21. Starting with version 10.0, JDK 11 is the preferred JDK version.

JDK	Recommended Version	Minimum Version
8	latest	u361
11	latest	11.0.18
17	latest	17.0.6
21	latest	21

The Tanzu GemFire product download does not include Java. Download and install a supported JRE or JDK on each system running Tanzu GemFire. VMware recommends the installation of a full JDK (and not just a JRE) to obtain better performance with gfsh status and gfsh stop commands.

The IBM SDK, Java Technology Edition, Version 8 is supported for application clients only. Some client region eviction configurations such as HEAP_LRU are known to not function properly with this SDK.

The following sections describe Java Modules and Tanzu GemFire.

How JDK 17 Encapsulation Affects Tanzu GemFire

Beginning with Java 17, the JVM strongly enforces the encapsulation policies introduced in Java 9 by the Java Platform Module System.

Several Tanzu GemFire features, such as serialization and eviction, use deep reflection to inspect the Java objects used as keys and values. In Java 17, this deep reflection requires that the object's type be made explicitly available for reflection.

By default, each type declared in a named module is available for reflection only if the declaring module opens the type's package.

As a result, Tanzu GemFire, by default, cannot inspect the following encapsulated types and fields:

- The private and protected types declared in named modules.
- The private and protected fields of types declared in named modules.

In Java 17, every type defined by the JDK is declared in a named module, and none are opened for reflection by any code outside of the JDK. Therefore, any encapsulated type or field defined by the JDK is, by default, unavailable for reflection.

See Access Required by Tanzu GemFire for ways to identify what access Tanzu GemFire needs, and Granting Access to Encapsulated Types for ways to grant that access.

Access Required by Tanzu GemFire

Tanzu GemFire may require reflective access to the types used in your application data.

If Tanzu GemFire does not have the access it requires, it throws an exception similar to the following:

```
java.lang.reflect.InaccessibleObjectException: Unable to make field private final
java.math.BigInteger java.math.BigDecimal.intVal accessible: module java.base does not
"opens java.math" to unnamed module @23a5fd2
```

Access to the Types Used in Application Data

Depending on the Tanzu GemFire features that you use, Tanzu GemFire may require reflective access to the types used in your application data. This includes:

- The types that define your application data.
- The types to which your application data holds a reference, whether directly or indirectly.

If any object of your application data is represented by a type declared in a named module, or holds a reference (directly or indirectly) to a type declared in a named module, Tanzu GemFire may require explicit access to that type.

> Every type defined by the JDK is declared in a named module. Therefore, if any object of your application data is represented by a type defined by the JDK, or holds a reference (directly or indirectly) to a type defined by the JDK, Tanzu GemFire may require explicit access to that type.

Tanzu GemFire can automatically access types declared in packages loaded from the classpath.

Additional Access Required by All Tanzu GemFire Processes

Tanzu GemFire always requires access to certain JDK packages, regardless of the types of objects used by your application. The following options must always be included when launching locators and servers, and when launching clients and other applications that depend on Tanzu GemFire:

```
--add-exports=java.base/sun.nio.ch=ALL-UNNAMED
--add-exports=java.management/com.sun.jmx.remote.security=ALL-UNNAMED
--add-opens=java.base/java.lang=ALL-UNNAMED
--add-opens=java.base/java.nio=ALL-UNNAMED
--add-opens=jdk.management/com.sun.management.internal=ALL-UNNAMED
```

> gfsh automatically supplies these options whenever it launches a locator or server process.

Granting Access to Encapsulated Types

Opening Specific Packages

To give Tanzu GemFire access to the types in a specific package, use the --add-opens command line option when launching a locator, server, client, or other application that uses Tanzu GemFire. For example:

```
--add-opens=java.base/java.math=ALL-UNNAMED
```

This example allows Tanzu GemFire to inspect BigDecimal and other types declared in the java.math package in the java.base module.

Because the JDK loads Tanzu GemFire code into an unnamed module, the --add-opens option must open the package to ALL-UNNAMED.

Using an Argument File

Starting with Java 9, the java command accepts argument files as an option. The java command treats the content of the argument file as arguments for the JVM.

If you must use numerous JVM options to give Tanzu GemFire the access it needs, you can combine the arguments into an argument file. For example:

```
--add-opens=java.base/java.text=ALL-UNNAMED
--add-opens=java.base/java.time=ALL-UNNAMED
--add-opens=java.base/java.time.chrono=ALL-UNNAMED
--add-opens=java.base/java.time.format=ALL-UNNAMED
--add-opens=java.base/java.time.temporal=ALL-UNNAMED
--add-opens=java.base/java.time.zone=ALL-UNNAMED
```

To use an argument file, add the @ prefix to identify your argument file to the java command:

```
java @path/to/my-argument-file ...
```

Opening All JDK Packages

The Tanzu GemFire distribution includes an argument file that opens every package in the Linux version of OpenJDK 17: path_to_product/config/open-all-jdk-packages-linux-openjdk-17.

Use the JDK 17 argument file as follows:

- When launching Tanzu GemFire clients and other applications with Tanzu GemFire functionality:

  ```
  java @path_to_product/config/open-all-jdk-packages-linux-openjdk-17 ...
  ```

- When launching Tanzu GemFire members with gfsh, add the argument file as a --J option:

  ```
  start locator --J=@path_to_product/config/open-all-jdk-packages-linux-openjdk-17 ...
  start server --J=@path_to_product/config/open-all-jdk-packages-linux-openjdk-17 ...
  ```

If you use a different JDK, copy and edit this file to add or remove packages to match your JDK.

Java Garbage Collection Defaults and Recommendations

On JDK 11 and later, the Garbage-First garbage collector (G1GC) is the Java default. However, VMware recommends using the Z garbage collector (ZGC) for most applications on JDK 17–20, and Generational ZGC on JDK 21 and later.

By default, gfsh enables the recommended garbage collector for your JVM version. The gfsh start locator and start server commands enable ZGC by default on JDK 17, and Generational ZGC on JDK 21. Recommended garbage collectors for each JDK version are summarized in the following table:

JDK Version	Java Default GC	Recommended GC	gfsh Default
JDK 8	Parallel	CMS	CMS
JDK 11	G1GC	CMS	CMS
JDK 17	G1GC	ZGC	ZGC
JDK 21	G1GC	Generational ZGC	Generational ZGC

> CMS, the Concurrent Mark Sweep collector is deprecated by the JDK in versions 8 and later. It is not available at all in JDKs 17 or 21.

Enable or Disable ZGC/Generational ZGC using gfsh

Use gfsh to configure the JVM garbage collector for your server or locator. By default, gfsh enables the recommended collector for your JVM version: ZGC on JDK 17–20 and Generational ZGC on JDK 21 and later. The following options are only required if you need to override this default.

- Enable ZGC (gfsh default on JDK 17–20):

  ```
  -XX:+UseZGC
  ```

- Disable ZGC and revert to G1GC:

  ```
  -XX:-UseZGC -XX:+UseG1GC
  ```

- Enable Generational ZGC (gfsh default on JDK 21 and later)
  To enable Generational ZGC, you must first enable ZGC

  ```
  -XX:+UseZGC -XX:+ZGenerational
  ```

- Disable Generational ZGC and revert to G1GC:

  ```
  -XX:-UseZGC -XX:-ZGenerational -XX:+UseG1GC
  ```

On a 64-bit JVM with a heap size less than 32 GB, switching from CMS or G1GC to ZGC may increase the cache's heap usage by up to 80%. The magnitude of this impact is highly dependent on the nature of your cached data.

On a 64-bit JVM with a heap size 32 GB or larger, switching from CMS or G1GC to ZGC does not increase the cache's heap usage.

On Linux, a JVM using ZGC may report a resident set size three times the actual memory usage.

For details about configuring Tanzu GemFire's use of the garbage collector, see Managing Heap Memory.

Supported Platforms and System Requirements for Tools

This topic links to documentation about supported platforms and system requirements for tools to use with VMware Tanzu GemFire.

- Pulse System Requirements

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

  > GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

- VSD System Requirements

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

Pulse System Requirements

This topic lists the installation and runtime requirements for VMware Tanzu GemFire Pulse. Verify that your system meets the these requirements before installation.

> Tanzu GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

The Pulse Web application has been tested for compatibility with the following Web browsers:

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

Pulse has been tested for standalone deployment on Tomcat and Jetty. Pulse may work with other operating systems and browsers upon which it has not been tested.

VSD System Requirements

This topic lists the Visual Statistics Display (VSD) requirements for use with Tanzu GemFire.

- VSD is supported on Linux, macOS, and Windows platforms.
- VSD is a 32-bit application. If you are running VSD on a 64-bit operating system, you might need to install 32-bit OS libraries to run the application if they are not already installed.
- On Linux, to find out which libraries are missing, run:

  ```
  ldd <product_dir>/tools/vsd/bin/vsdwishLinux
  ```

- To use VSD on Windows 7 and later, perform the following steps:
  1. Start Windows Explorer and navigate to the `GEMFIREPRODUCTDIR\tools\vsd\bin\` directory, where `GEMFIREPRODUCTDIR` is 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 for all the other executables in the `bin` directory.

For more information, see Installing and Running VSD.

Getting Started with Tanzu GemFire

This topic provides links to documentation that describes key functionality of VMware Tanzu GemFire and a tutorial to demonstrate features.

See the gemfire-examples github repository for examples showing how to use GemFire in a variety of use cases.

- About VMware Tanzu GemFire

  Tanzu GemFire is a data management platform that provides real-time, consistent access to data-intensive applications throughout widely distributed cloud architectures.

- Main Features

  This section summarizes the main features and key functionality of VMware Tanzu GemFire.

- Installing Tanzu GemFire
- Upgrading Tanzu GemFire
- Tanzu GemFire in 15 Minutes or Less

  Take this brief tour to try out basic features and functionality.

About Tanzu GemFire

VMware Tanzu GemFire is an enterprise grade, high speed in-memory data and compute grid for high-performance, low latency applications where data must be consistent, up to date and delivered with sub-millisecond response times at scale, across multiple data centers.

Tanzu 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, high performance, scalability, and fault tolerance. In addition to being a distributed data container, Tanzu 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 Tanzu GemFire distributed system. Application architects can arrange these nodes in peer-to-peer or client/server 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 from a region.

Tanzu GemFire *locators* provide both member discovery and load-balancing services. You configure clients with a list of locator services and the locators maintain a dynamic list of member servers.

Tanzu GemFire uses *continuous querying* to enable event-driven architectures. Tanzu 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.

In addition to peer-to-peer and client/server topologies, Tanzu GemFire supports multi-site configurations that allow you to scale horizontally between disparate, loosely-coupled distributed systems over a wide geographically separated network. A wide-area network (WAN) is the main use case for the multi-site topology.

Main Features

This topic summarizes the main features and key functionality of VMware Tanzu GemFire.

- High Read-and-Write Throughput
- Low and Predictable Latency
- High Scalability
- Continuous Availability
- Reliable Event Notifications
- Parallelized Application Behavior on Data Stores
- Shared-Nothing Disk Persistence
- Reduced Cost of Ownership
- Single-Hop Capability for Client/Server
- Client/Server Security
- Multisite Data Distribution
- Continuous Querying
- Heterogeneous Data Sharing

High Read-and-Write Throughput

Read-and-write throughput is provided by concurrent main-memory data structures and a highly optimized distribution infrastructure. Applications can make copies of data dynamically in memory through synchronous or asynchronous replication for high read throughput or partition the data across many 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

The 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.

Subscription management (interest registration and continuous queries) is partitioned 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

Scalability is achieved 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. This 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 members synchronously or asynchronously by using a “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 requires them to access a standard database when the event is delivered, limiting the subscriber by the speed of the database.

Data and events are offered 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.

Parallelized Application Behavior on Data Stores

You can execute application business logic in parallel on members. The 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 colocating the relevant data and parallelizing the calculation, you increase overall throughput. 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 function and to avoid moving data around on the network. Application function can be executed on only one member, 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 cluster 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, separate files are not used to manage data and transaction logs. 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.

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 directly to the server holding the data key, avoiding multiple hops to locate data that is partitioned. Metadata in the client identifies the correct server. This feature improves performance and client access to partitioned regions in the server tier.

Client/Server Security

There may be multiple, distinct users in client applications. This feature accommodates installations in which 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 has its own access authorization to the server cache.

Multisite Data Distribution

Scalability problems can result from data sites being spread out geographically across a wide-area network (WAN). Models address these topologies, ranging from a single peer-to-peer cluster to reliable communications between data centers across the WAN. This model allows clusters 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 an increase in 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.

Continuous Querying

In messaging systems like Java Message Service, clients subscribe to topics and queues. Any message delivered to a topic is sent to the subscriber. Tanzu GemFire allows continuous querying by having applications express complex interest using Object Query Language.

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.

Installing Tanzu GemFire

This topic explains how to install Tanzu GemFire and provides links to additional information.

When you install a new version of Tanzu GemFire software on an existing Tanzu GemFire system, keep the previous version available until the system upgrade has completed and been verified.

- Installing Tanzu GemFire

  You can install Tanzu GemFire in the following ways:

  - Option 1: Install Tanzu GemFire from a compressed TAR File
  - Option 2: Install Tanzu GemFire on the Elastic Application Runtime
  - Option 3: Install Tanzu GemFire on Tanzu Platform for Kubernetes
  - Option 4: Install Tanzu GemFire from Docker Hub
- Installing VMware Tanzu GemFire Extensions

  You can add Tanzu GemFire packages that enhance the functionality of the core platform.
- Uninstalling Tanzu GemFire

  This topic describes how to remove Tanzu GemFire.

Obtaining and Installing Security Updates

New versions of Tanzu GemFire often include important security fixes. VMware recommends that you stay current with the latest releases.

For details about any security fixes in a particular release, see VMware Tanzu Security.

Installing Tanzu GemFire from a Compressed TAR File

This topic explains how to install VMware Tanzu GemFire from a compressed TAR file on Windows, Unix, and Linux.

Use the compressed TAR file distribution to install and configure Tanzu GemFire on every physical and virtual machine where you will run Tanzu GemFire.

Prerequisites

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

- Confirm that your system meets the hardware and software requirements described in Supported Configurations and System Requirements.
- 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.)

Complete the following steps to download:

1. Log in to the Broadcom Customer Support Portal with your customer credentials. Before downloading, ensure you meet the requirements, for more information, see the Download Broadcom products and software article.
2. Go to the VMware Tanzu GemFire downloads page. Select VMware Tanzu GemFire, click Show All Releases, and select a version.
3. Review the Terms and Conditions, and then click I agree to Terms and Conditions.
4. Click the HTTPS Download icon next to VMware GemFire to download.

Procedure

Use the following procedure to install Tanzu GemFire:

1. Navigate to the directory where you downloaded the GemFire software, and expand the compressed TAR file after creating the *path_to_product* directory.

   ```
   $ tar -xzvf vmware-gemfire-N.N.N.tgz -C path_to_product
   ```

   path_to_product corresponds to the location where you want to install GemFire, and *N.N.N* is the version number.

2. Configure the JAVA_HOME environment variable.

   If you will be using the gfsh command-line utility then you must set JAVA_HOME to a JDK installation. For example:

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

     ```
     JAVA_HOME=/usr/java/jdk1.8.0_361
     export JAVA_HOME
     ```

   - Windows

     ```
     set JAVA_HOME="C:\Program Files\Java\jdk1.8.0_361"
     ```

3. This step only applies to environments where you are running GemFire processes or GemFire client applications outside of gfsh. The gfsh script sets these environment variables for you. If you are running GemFire processes or applications outside of gfsh, then configure the following environment variables for GemFire.

   - Set the GEMFIRE_HOME environment variable to point to your GemFire installation top-level directory. (You should see bin, lib, dtd, and other directories under GEMFIRE_HOME.) The following variables definitions are examples; your installation

path will vary, depending on where you install GemFire and the version (*N.N.N*) you are installing.

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

```
GEMFIRE_HOME=/opt/vmware/vmware-gemfire-N.N.N
export GEMFIRE_HOME
```

- Windows

```
set GEMFIRE_HOME=C:\vmware\gemfire\vmware-gemfire-N.N.N
```

- Configure your 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.)
  - UNIX and Linux (Bourne and Korn shells - sh, ksh, bash)

```
GF_JAVA=$JAVA_HOME/bin/java
export GF_JAVA
```

  - Windows

```
set GF_JAVA=%JAVA_HOME%\bin\java.exe
```

4. Add GemFire scripts to your the PATH environment variable. For example:
   - UNIX and Linux (Bourne and Korn shells - sh, ksh, bash)

```
PATH=$PATH:$JAVA_HOME/bin:/opt/vmware/vmware-gemfire-N.N.N/bin
export PATH
```

   - Windows

```
set PATH=%PATH%;%JAVA_HOME%\bin;%GEMFIRE_HOME%\bin
```

5. Type `gfsh version` at the command line and verify that the output lists the version of Tanzu GemFire that you wish to install. For example:

```
$ gfsh version
v10.1.0
```

   If you want more detailed version information such as the date of the build, build number and JDK version being used, type `gfsh version --full`.

6. Repeat this procedure for every virtual or physical machine on which you will run Tanzu GemFire.

Installing Tanzu GemFire Extensions

This topic explains how to install VMware Tanzu GemFire packages, or *extensions*, to Tanzu GemFire.

These extensions enhance the functionality of the core platform and are packaged in .gfm files that load on Tanzu GemFire startup. Examples of extensions include the Tanzu GemFire Session Management and Tanzu GemFire Vector Database.

Each Tanzu GemFire extension provides specific details for its installation. Typically, you install an extension by loading its .gfm file into the `${GEMFIRE_HOME}/extensions` directory, where Tanzu GemFire searches to discover extensions on startup.

You can specify different extension discovery locations by defining the variable GEMFIRE_EXTENSIONS_REPOSITORY_PATH in the environment in which Tanzu GemFire executes. Specify multiple locations using the syntax that the execution environment provides for defining PATH variables. For example, in Linux systems, specify the locations as a colon-separated list. For Windows, separate the locations by semicolons.

> If you wish to retain ${GEMFIRE_HOME}/extensions as one of the options, you must include it in the list when you define the GEMFIRE_EXTENSIONS_REPOSITORY_PATH environment variable.

Refer to the documentation for each Tanzu GemFire extension for specific installation details.

Uninstalling Tanzu GemFire

This topic explains how to uninstall VMware Tanzu GemFire.

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

If the old version of Tanzu GemFire was installed with an RPM, uninstall using the following command:

```
rpm -e Pivotal_GemFire_XXX
```

Where XXX is the Tanzu GemFire version number and also corresponds to the name of the product installation directory. As an example, the command for removing the Tanzu GemFire 8.2.5 release would be as follows:

```
rpm -e Pivotal_GemFire_825
```

If the old version of Tanzu GemFire was installed with DEBs, uninstall using the following command:

```
dpkg --remove pivotal-gemfire
```

If you installed Tanzu GemFire by using macOS brew packages, issue the following command:

```
brew uninstall gemfire
```

Upgrading Tanzu GemFire

This topic describes how to plan a VMware Tanzu GemFire upgrade.

To upgrade an existing installation to a new version of Tanzu GemFire, follow these general steps:

1. Back up your current system.
2. Install the new version of the software.
3. Stop your cluster using the current software.
4. Restart the system using the new software.

In many cases, components running under the current version can be stopped selectively, then restarted under the new version so that the cluster as a whole remains functional during the upgrade process; this is known as a “rolling upgrade.”

In other cases, the entire system must be stopped in order to accomplish the upgrade, which will require some downtime for your system.

See Planning an Upgrade to choose the upgrade scenario that best suits your implementation and to understand the resources you will need to accomplish the upgrade. Then select the appropriate upgrade procedure for more detailed instructions that fit your specific needs.

Upgrade Details

- Planning an Upgrade

  This section discusses the upgrade paths for various VMware Tanzu GemFire versions, and it lists information you need to know before you begin the upgrade process.

- Rolling Upgrade

  A rolling upgrade allows you to keep your existing cluster running while you upgrade your members one at a time.

- Offline Upgrade

  An offline upgrade can handle the widest variety of software versions and cluster configurations, but requires shutting down the entire system for at least a short time.

- Upgrading Clients

  When you upgrade your GemFire server software, you may need to update your client applications in order to maintain compatibility with the upgraded servers.

- Upgrading GemFire to Version 10.x

  Items of particular interest to those upgrading to GemFire version 10 from version 9.

Planning an Upgrade

This topic explains how to plan an upgrade of VMware Tanzu GemFire.

Before you upgrade your system, back it up. Make backup copies of all existing disk-stores, server-side code, configuration files, and data across the entire cluster. To get a backup of the data that includes the most recent changes may require that traffic across the cluster is stopped before the backup is made.

For a description of the backup process, see Creating Backups for System Recovery and Operational Management.

For information about running the `gfsh backup disk-store` command, see the backup disk-store command reference page.

> When connected to a GemFire version 10.1 or later locator, if members running a GemFire version earlier than 10.1 exist in the cluster, issuing the `backup disk-store` command results in the following error message: “Backup only supports GemFire 10.1 and later. The following members are running on an earlier GemFire version.” When connected to a GemFire version 10.0 or earlier locator, the `backup disk-store` runs successfully even if other locators have been upgraded to 10.1 or later.

Guidelines for Upgrading

- Schedule your upgrade during a period of low user activity for your system and network.

- Verify that the machines hosting the cluster members meet the Host Machine Requirements of the upgraded cluster.
- Important: After all locators have been upgraded, *do not start or restart any processes* that use the older version of the software. The older process will either not be allowed to join the distributed system or, if allowed to join, can potentially cause a deadlock.
- Verify that all members that you wish to upgrade are members of the same distributed system cluster. A list of cluster members will be output with the gfsh command:

  ```
  gfsh>list members
  ```

- Locate a copy of your system's startup script, if your site has one (most do). The startup script can be a handy reference for restarting upgraded locators and servers with the same gfsh command lines that were used in your current installation.
- Identify how your current cluster configuration was specified. The way in which your cluster configuration was created determines which commands you use to save and restore that cluster configuration during the upgrade procedure. There are two possibilites:
  - With gfsh commands, relying on the underlying cluster configuration service to record the configuration: see Exporting and Importing Cluster Configurations.
  - With XML properties specified through the Java API or configuration files: see Deploying Configuration Files without the Cluster Configuration Service.
- Do not modify region attributes or data, either via gfsh or cache.xml configuration, during the upgrade process.

Version Compatibilities

- Before you upgrade GemFire to a newer version, confirm that your system meets the hardware and software requirements for the newer version described in Supported Configurations and System Requirements.
- For best reliability and performance, all server components of a Tanzu GemFire system should run the same version of the software. For the purposes of a rolling upgrade, you can have peers or cache servers running different minor versions of VMware Tanzu GemFire at the same time, as long as the major version is the same. For example, some components can continue to run under version 9.10 while you are in the process of upgrading to version 9.15.
- It is not possible to directly upgrade from a GemFire version earlier than 10.0.0 to GemFire v10.2.0 or later. To upgrade from GemFire 9.x to 10.2, a two-stepped upgrade is required, the process is required, for more information, see Upgrading VMware Tanzu GemFire from 9.x to Version 10.2.

Upgrade to a Newer Version

VMware recommends that you upgrade to the newest version of Tanzu GemFire. Upgrade to a newer version in one of the following ways:

- Rolling Upgrade: If possible, follow the Rolling Upgrade procedure. With a multi-site installation, you can perform a rolling upgrades within each site.
- Off-Line Upgrade: If you cannot perform a rolling upgrade, follow the Off-Line Upgrade procedure.

> You cannot perform a rolling upgrade on a cluster that has partitioned regions without redundancy. Without the redundancy, region entries will be lost when individual servers are taken out of the cluster during a rolling upgrade.

Rolling Upgrade

This topic explains how to perform a rolling upgrade of VMware Tanzu GemFire.

A rolling upgrade eliminates system downtime by keeping your existing distributed system running while you upgrade one member at a time. Each upgraded member can communicate with other members that are still running the earlier version of Tanzu GemFire, so servers can respond to client requests even as the upgrade is underway. Interdependent data members can be stopped and started without mutually blocking, a problem that can occur when multiple data members are stopped at the same time.

Rolling Upgrade Limitations and Requirements

Versions

Rolling upgrade requires that the older and newer versions of Tanzu GemFire are mutually compatible, which usually means that they share the same major version number. Therefore, you can perform a rolling upgrade to upgrade from earlier versions of 9.x up to the most recent version of 10.2.

Components

Rolling upgrades apply to the peer members or cache servers within a distributed system. Under some circumstances, rolling upgrades can also be applied within individual sites of multi-site (WAN) deployments.

Redundancy

All partitioned regions in your system must have full redundancy. Check the redundancy state of all your regions *before* you begin the rolling upgrade and *before* stopping any members. See Checking Redundancy in Partitioned Regions for details.

If a rolling update is not possible for your system, follow the Off-Line Upgrade procedure.

Rolling Upgrade Guidelines

Do not create or destroy regions

When you perform a rolling upgrade, your online cluster will have a mix of members running different versions of GemFire. During this time period, do not execute region operations such as region creation or region destruction.

Region rebalancing affects the restart process

If you have startup-recovery-delay deactivated (set to -1) for your partitioned region, you must perform a rebalance on your region after you restart each member. If rebalance occurs automatically, as it will if startup-recovery-delay is enabled (set to a value other than -1), make sure that the rebalance completes before you stop the next server. If you have startup-recovery-delay enabled and set to a high number, you may need to wait extra time until the region has recovered redundancy, because rebalance must complete before new servers are restarted. The partitioned region attribute startup-recovery-delay is described in Configure Member Join Redundancy Recovery for a Partitioned Region.

Checking component versions while upgrading

During a rolling upgrade, you can check the current GemFire version of all members in the cluster by looking at the server or locator logs.

When an upgraded member reconnects to the distributed system, it logs all the members it can see as well as the GemFire version of those members. For example, an upgraded locator will now detect GemFire members running the older version of GemFire (in this case, the version being upgraded: GFE 9.0.0) :

```
[info 2013/06/03 10:03:29.206 PDT frodo <vm_1_thr_1_frodo> tid=0x1a]
DistributionManager frodo(locator1:21869:locator)<v16>:28242 started
on frodo[15001]. There were 2 other DMs. others:
[frodo(server2:21617)<v4>:14973( version:GFE 9.0.0 ),
frodo(server1:21069)<v1>:60929( version:GFE 9.0.0 )] (locator)
```

After some members have been upgraded, non-upgraded members will log the following message when they receive a new membership view:

```
Membership: received new view [frodo(locator1:20786)<v0>:32240|4]
  [frodo(locator1:20786)<v0>:32240/51878,
  frodo(server1:21069)<v1>:60929/46949,
  frodo(server2:21617)<v4>( version:UNKNOWN[ordinal=23] ):14973/33919]
```

Non-upgraded members identify members that have been upgraded to the next version with version: UNKNOWN.

Cluster configuration affects save and restore

The way in which your cluster configuration was created determines which commands you use to save and restore that cluster configuration during the upgrade procedure.

- If your system was configured with gfsh commands, relying on the underlying cluster configuration service, the configuration can be saved in one central location, then applied to all newly-upgraded members. See Exporting and Importing Cluster Configurations.
- If your system was configured with XML properties specified through the Java API or configuration files, you must save the configuration for each member before you bring it down, then re-import it for that member's upgraded counterpart. See Deploying Configuration Files without the Cluster Configuration Service.

Rolling Upgrade Procedure

Begin by installing the new version of the software alongside the older version of the software on all hosts. You will need both versions of the software during the upgrade procedure. See Installing Tanzu GemFire.

Upgrade locators first, then data members, then clients.

Upgrade Locators

1. On the machine hosting the first locator you wish to upgrade, open a terminal console.
2. Start a gfsh prompt, using the version from your current GemFire installation, and connect to the currently running locator. For example:

   ```
   gfsh>connect --locator=locator_hostname_or_ip_address[port]
   ```

3. Use gfsh commands to characterize your current installation so you can compare your post-upgrade system to the current one. For example, use the list members command to view locators and data members:

```
 Name   | Id
-------- | ----------------------------------------------
locator1 | 172.16.71.1(locator1:26510:locator)<ec><v0>:1024
locator2 | 172.16.71.1(locator2:26511:locator)<ec><v1>:1025
server1  | 172.16.71.1(server1:26514)<v2>:1026
server2  | 172.16.71.1(server2:26518)<v3>:1027
```

4. Save your cluster configuration.
   - If you are using the cluster configuration service, use the gfsh export cluster-configuration command. You only need to do this once, as the newly-upgraded locator will propagate the configuration to newly-upgraded members as they come online.
   - For an XML configuration, save cache.xml, gemfire.properties, and any other relevant configuration files to a well-known location. You must repeat this step for each member you upgrade.
5. Stop the locator. For example:

```
gfsh>stop locator --name=locator1
Stopping Locator running in /Users/username/sandbox/locator
on 172.16.71.1[10334] as locator...
Process ID: 96686
Log File: /Users/username/sandbox/locator/locator.log
....
No longer connected to 172.16.71.1[1099].
```

6. Start gfsh from the new GemFire installation. Verify that you are running the newer version with

```
gfsh>version
```

7. Start a locator and import the saved configuration. If you are using the cluster configuration service, use the same name and directory as the older version you stopped, and the new locator will access the old locator's cluster configuration without having to import it in a separate step:

```
gfsh>start locator --name=locator1 --enable-cluster-configuration=true \
--dir=/data/locator1
```

   Otherwise, use the gfsh import cluster-configuration command or explicitly import .xml and .properties files, as appropriate.

8. The new locator should reconnect to the same members as the older locator. Use list members to verify:

```
gfsh>list members
 Name   | Id
-------- | ----------------------------------------------------
locator1 | 172.16.71.1(locator1:26752:locator)<ec><v17>:1024(version:UNKNOWN[ordinal=65])
locator2 | 172.16.71.1(locator2:26511:locator)<ec><v1>:1025
server1  | 172.16.71.1(server1:26514)<v2>:1026
server2  | 172.16.71.1(server2:26518)<v3>:1027
```

9. Upgrade the remaining locators by stopping and restarting them. When you have completed that step, the system gives a more coherent view of version numbers:

```
gfsh>list members
  Name   | Id
-------- | ----------------------------------------------------
locator1 | 172.16.71.1(locator1:26752:locator)<ec><v17>:1024
locator2 | 172.16.71.1(locator2:26808:locator)<ec><v30>:1025
server1  | 172.16.71.1(server1:26514)<v2>:1026(version:GFE 9.0)
server2  | 172.16.71.1(server2:26518)<v3>:1027(version:GFE 9.0)
```

The server entries show that the servers are running an older version of gemfire, in this case (version:GFE 9.0).

Upgrade Servers

After you have upgraded all of the system's locators, upgrade the servers.

1. Upgrade each server, one at a time, by stopping it and restarting it. Restart the server with the same command-line options with which it was originally started in the previous installation. For example:

   ```
   gfsh>stop server --name=server1
   Stopping Cache Server running in /Users/share/server1
   on 172.16.71.1[52139] as server1...

   gfsh>start server --name=server1 --use-cluster-configuration=true \
   --server-port=0 --dir=/data/server1
   Starting a Geode Server in /Users/share/server1...
   ```

   Use the list members command to verify that the server is now running the new version of GemFire:

   ```
   gfsh>list members
     Name   | Id
   -------- | ----------------------------------------------------
   locator1 | 172.16.71.1(locator1:26752:locator)<ec><v17>:1024
   locator2 | 172.16.71.1(locator2:26808:locator)<ec><v30>:1025
   server1  | 172.16.71.1(server1:26835)<v32>:1026
   server2  | 172.16.71.1(server2:26518)<v3>:1027(version:GFE 9.0)
   ```

2. Restore data to the data member. If automatic rebalancing is enabled (partitioned region attribute startup-recovery-delay is set to a value other than -1), data restoration will start automatically. If automatic rebalancing is deactivated (partitioned region attribute startup-recovery-delay=-1), you must initiate data restoration by issuing the gfsh rebalance command.

   Wait until the newly-started server has been restored before upgrading the next server. You can repeat various gfsh show metrics command with the --member option or the --region option to verify that the data member is hosting data and that the amount of data it is hosting has stabilized.

3. Shut down,restart, and rebalance servers until all data members are running the new version of GemFire.

Upgrade Clients

Upgrade Tanzu GemFire clients, following the guidelines described in Upgrading Clients.

Offline Upgrade

This topic explains how to upgrade VMware Tanzu GemFire when you cannot or choose not to perform a rolling upgrade.

For example, a rolling upgrade is not possible for a cluster that has partitioned regions without redundancy. Without the redundancy, region entries would be lost when individual servers were taken out of the cluster during a rolling upgrade.

Offline Upgrade Guidelines

Versions

For best reliability and performance, all server components of a Tanzu GemFire system should run the same version of the software. See Version Compatibilities for more details on how different versions of Tanzu GemFire can interoperate.

Data member interdependencies

When you restart your upgraded servers, interdependent data members may hang on startup waiting for each other. In this case, start the servers in separate command shells so they can start simultaneously and communicate with one another to resolve dependencies.

Offline Upgrade Procedure

1. Stop any connected clients.
2. On a machine hosting a locator, open a terminal console.
3. Start a gfsh prompt, using the version from your current Tanzu GemFire installation, and connect to a currently running locator. For example:

   ```
   gfsh>connect --locator=locator_hostname_or_ip_address[port]
   ```

4. Use gfsh commands to characterize your current installation so you can compare your post-upgrade system to the current one. For example, use the list members command to view locators and data members:

   ```
   gfsh>list members
   Name      | Id                                                                | Type
   --------- | ------------------------------------------------------------------| ---——---
   locator-0 | 192.168.43.99(locator-0:16523:locator)<ec><v0>:42586 [Coordinator] | Locator
   locator-1 | 192.168.43.99(locator-1:16524:locator)<ec><v1>:42863               | Locator
   server-2  | 192.168.43.99(server-2:16525)<v2>:47137                            | Server
   server-1  | 192.168.43.99(server-1:16526)<v3>:44209                            | Server
   ```

5. Save your cluster configuration.
   - If you are using the cluster configuration service, use the gfsh export cluster-configuration command. You only need to do this once, as the newly-upgraded locator will propagate the configuration to newly-upgraded members as they come online.
   - For an XML configuration, save cache.xml, gemfire.properties, and any other relevant configuration files to a well-known location. You must repeat this step for each member you upgrade.
6. Shut down the entire cluster (by pressing Y at the prompt, this will lose no persisted data):

```
gfsh>shutdown --include-locators=true
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
Shutdown is triggered

gfsh>
No longer connected to 172.16.71.1[1099].
gfsh>quit
```

Since Tanzu GemFire is a Java process, to check before continuing that all Tanzu GemFire members successfully stopped, it is useful to use the JDK-included jps command to check running java processes:

```
% jps
29664 Jps
```

7. On each machine in the cluster, install the new version of the software (alongside the older version of the software). See Installing Tanzu GemFire.
8. Redeploy your environment's configuration files to the new version installation. If you are using the cluster configuration service, one copy of the exported .zip configuration file is sufficient, as the first upgraded locator will propagate it to the other members. For XML configurations, you should have a copy of the saved configuration files for each data member.
9. On each machine in the cluster, install any updated server code. Point all client applications to the new installation of Tanzu GemFire.
10. Run the new version of gfsh.
11. Start a locator and import the saved configuration. If you are using the cluster configuration service, use the same name and directory as the older version you stopped, and the new locator will access the old locator's cluster configuration without having to import it in a separate step:

    ```
    gfsh>start locator --name=locator1 --enable-cluster-configuration=true \
    --dir=/data/locator1
    ```

    Otherwise, use the gfsh import cluster-configuration command or explicitly import .xml and .properties files, as appropriate.
12. Restart all system data members using the new version of gfsh. Use the same names, directories, and other properties that were used when starting the system under the previous version of the software. (Here is where a system startup script comes in handy as a reference.) Interdependent data members may hang on startup waiting for each other. In this case, start servers in separate shells so they can communicate with one another to resolve dependencies.
13. Upgrade Tanzu GemFire clients, following the guidelines described in Upgrading Clients.

Upgrading Clients

This topic explains how to upgrade clients when you upgrade your VMware Tanzu GemFire server software.

When you upgrade your Tanzu GemFire server software, you may need to update your client applications to maintain compatibility with the upgraded servers. To support real-world implementations, servers can usually interoperate with a few different versions of the client software. In general, you will have best performance and reliability if:

- All clients run the same version of the client software.
- Clients and servers both run the latest versions of their respective software.

Tanzu GemFire server and native client software releases follow similar numbering schemes, but they are not released in lockstep. See the Tanzu GemFire Session Management for more detailed information regarding version compatibility.

Changes you may need to make when you update, recompile, and link your client code include:

- Removing or replacing obsolete identifiers
- Renaming packages
- Reinstating secure client/server messaging

Remove or Replace Obsolete Identifiers

Review the Release Notes for a list of classes, methods, and other identifiers that are no longer present in the current release. Update client code so it no longer uses any of these removed identifiers.

Protocol Alignment

Due to a protocol alignment/client-server handshake fix in GemFire v9.15 (GEODE-9982, GEM-3509), GemFire Native Client applications must be upgraded to at least version 10.2.7 before upgrading to GemFire 9.15.

Upgrading from 9.x to 10.2

This topic explains how to upgrade VMware Tanzu GemFire from version 9.x to version 10.2.

> GemFire v10.2 removes the JGroups library that previously enabled UDP-based communication. As older GemFire versions (prior to v10.0.0) require UDP, you cannot upgrade directly from GemFire 9.x to GemFire v10.2.0 or later, a two-step upgrade is required. To complete this upgrade, you must first upgrade to v10.0 or v10.1.

Procedure

1. Upgrade to GemFire v10.0 or v10.1

   If possible, perform a rolling upgrade; if not, perform an off-line upgrade. For more information, see:

   - Planning an Upgrade to v10.0
   - Planning an Upgrade to v10.1

2. Update configuration properties

   Before upgrading to version 10.2, ensure the following configuration properties are correctly set:

   - disable-tcp is either unspecified or set to false
   - use-udp-membership-messenger is either unspecified or set to false

3. Upgrade to GemFire version 10.2.0 or later

If possible, perform a rolling upgrade; if not, perform an off-line upgrade. For more information, see Planning an Upgrade.

Tanzu GemFire in 15 Minutes or Less

Do you need a quick introduction to VMware Tanzu GemFire? Take this brief tour to try out basic features and functionality.

Step 1. Install Tanzu GemFire

See Installing Tanzu GemFire for instructions.

Step 2. Use gfsh to start a locator

In a terminal window, use the gfsh command line interface to start up a locator. *gfsh* (pronounced "jee-fish") provides a single, intuitive command-line interface from which you can launch, manage, and monitor Tanzu GemFire processes, data, and applications. See gfsh.

The *locator* is a GemFire process that tells new, connecting members where running members are located and provides load balancing for server use. A locator, by default, starts up a JMX Manager, which is used for monitoring and managing a GemFire cluster. The cluster configuration service uses locators to persist and distribute cluster configurations to cluster members. See Running GemFire Locator Processes and Overview of the Cluster Configuration Service.

1. Create a scratch working directory (for example, my_gemfire) and change directories into it. gfsh saves 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).

```
  _________________________     __
 / _____/ _______/ _______/ /____/ /
 / /  __/ /___  /_______  / _______  /
 / /__/ / ____/  _______/ / /    / /
/_______/_/      /_______/_/    /_/

Monitor and Manage GemFire
gfsh>
```

3. At the gfsh prompt, type the start locator command and specify a name for the locator:

```
gfsh>start locator --name=locator1
Starting a GemFire Locator in /home/username/my_gemfire/locator1...
.................................
Locator in /home/username/my_gemfire/locator1 on ubuntu.local[10334] as locator1 is currently online.
Process ID: 3529
Uptime: 18 seconds
GemFire Version: 10.1.0
Java Version: 11.0.17
Log File: /home/username/my_gemfire/locator1/locator1.log
JVM Arguments: -Dgemfire.enable-cluster-configuration=true -Dgemfire.load-cluster-configuration-from-dir=false
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /home/username/gf10/lib/gemfire-bootstrap-10.1.0.jar

Successfully connected to: JMX Manager [host=ubuntu.local, port=1099]

Cluster configuration service is up and running.
```

If you run start locator from gfsh without specifying the member name, gfsh will automatically pick a random member name. This is useful for automation.

Step 3. Start Pulse

Start up the browser-based Pulse monitoring tool. Pulse is a Web Application that provides a graphical dashboard for monitoring vital, real-time health and performance of GemFire clusters, members, and regions. See Tanzu GemFire Pulse.

> GemFire 10.1 deprecates Pulse in favor of Tanzu GemFire Management Console. Pulse will be removed in a future release.

```
gfsh>start pulse
Launched GemFire 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.

The Pulse application now displays the locator you just started (locator1):

Step 4. Start a server

A GemFire server is a Tanzu GemFire process that runs as a long-lived, configurable member of a cluster. 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. See Running GemFire Server Processes.

Start the cache server:

```
gfsh>start server --name=server1 --server-port=40411
```

This commands starts a cache server named “server1” on the specified port of 40411.

If you run start server from gfsh without specifying the member name, gfsh will automatically pick a random member name. This is useful for automation.

Observe the changes (new member and server) in Pulse. Try expanding the distributed system icon to see the locator and cache server graphically.

Step 5. Create a replicated, persistent region

In this step you create a region with the gfsh command line utility. Regions are the core building blocks of the GemFire cluster and provide the means for organizing your data. The region you create for this exercise employs replication to replicate data across members of the cluster and utilizes persistence to save the data to disk. See Data Regions.

1. Create a replicated, persistent region:

   ```
   gfsh>create region --name=regionA --type=REPLICATE_PERSISTENT
   Member  | Status
   ------- | --------------------------------------
   server1 | Region "/regionA" created on "server1"
   ```

   Note that the region is hosted on server1.

2. Use the gfsh command line to view a list of the regions in the cluster.

   ```
   gfsh>list regions
   List of regions
   ---------------
   regionA
   ```

3. List the members of your cluster. The locator and cache server you started appear in the list:

   ```
   gfsh>list members
   Name      | Id                                                                | Type
   --------- | ------------------------------------------------------------------| ---———---
   locator-0 | 192.168.43.99(locator-0:16523:locator)<ec><v0>:42586 [Coordinator] | Locator
   locator-1 | 192.168.43.99(locator-1:16524:locator)<ec><v1>:42863               | Locator
   server-1  | 192.168.43.99(server-1:16526)<v3>:44209                            | Server
   ```

4. To view specifics about a region, type the following:

   ```
   gfsh>describe region --name=regionA
   ..........................................................
   Name            : regionA
   Data Policy     : persistent replicate
   Hosting Members : server1

   Non-Default Attributes Shared By Hosting Members

    Type  | Name | Value
   ------ | ---- | -----
   Region | size | 0
   ```

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

Keep this gfsh prompt open for the next steps.

Step 6. Manipulate data in the region and demonstrate persistence

Tanzu GemFire manages data as key/value pairs. In most applications, a Java program adds, deletes and modifies stored data. You can also use gfsh commands to add and retrieve data. See Data Commands

1. Run the following put commands to add some data to the region:

```
gfsh>put --region=regionA --key="1" --value="one"
Result      : true
Key Class   : java.lang.String
Key         : 1
Value Class : java.lang.String
Old Value   : <NULL>


gfsh>put --region=regionA --key="2" --value="two"
Result      : true
Key Class   : java.lang.String
Key         : 2
Value Class : java.lang.String
Old Value   : <NULL>
```

2. Run the following command to retrieve data from the region:

```
gfsh>query --query="select * from /regionA"

Result     : true
startCount : 0
endCount   : 20
Rows       : 2

Result
------
two
one
```

   Note that the result displays the values for the two data entries you created with the put commands.

   See Data Entries.

3. Stop the cache server using the following command:

```
gfsh>stop server --name=server1
Stopping Cache Server running in /home/username/my_gemfire/server1 on ubuntu.local[40411] as server1...
Process ID: 3883
Log File: /home/username/my_gemfire/server1/server1.log
....
```

4. Restart the cache server using the following command:

```
gfsh>start server --name=server1 --server-port=40411
```

5. Run the following command to retrieve data from the region again – notice that the data is still available:

```
gfsh>query --query="select * from /regionA"

Result     : true
startCount : 0
endCount   : 20
Rows       : 2

Result
------
two
one
```

Because regionA uses persistence, it writes a copy of the data to disk. When a server hosting regionA starts, the data is populated into the cache. Note that the result displays the values for the two data entries you created with the put commands prior to stopping the server.

See Data Entries.

See Data Regions.

Step 7. Examine the effects of replication

In this step, you start a second cache server. Because regionA is replicated, the data will be available on any server hosting the region.

1. Start a second cache server:

```
gfsh>start server --name=server2 --server-port=40412
```

2. Run the describe region command to view information about regionA:

```
gfsh>describe region --name=regionA
.......................................................
Name            : regionA
Data Policy     : persistent replicate
Hosting Members : server1
                  server2

Non-Default Attributes Shared By Hosting Members

 Type  | Name | Value
------ | ---- | -----
Region | size | 2
```

Note that you do not need to create regionA again for server2. The output of the command shows that regionA is hosted on both server1 and server2. When gfsh starts a server, it requests the configuration from the cluster configuration service which then distributes the shared configuration to any new servers joining the cluster.

3. Add a third data entry:

```
gfsh>put --region=regionA --key="3" --value="three"
Result      : true
Key Class   : java.lang.String
Key         : 3
Value Class : java.lang.String
Old Value   : <NULL>
```

4. Open the Pulse application (in a Web browser) and observe the cluster topology. You should see a locator with two attached servers. Click the Data tab to view information about regionA.
5. Stop the first cache server with the following command:

```
gfsh>stop server --name=server1
Stopping Cache Server running in /home/username/my_gemfire/server1 on ubuntu.local[40411] as server1...
Process ID: 4064
Log File: /home/username/my_gemfire/server1/server1.log
....
```

6. Retrieve data from the remaining cache server.

```
gfsh>query --query="select * from /regionA"

Result     : true
startCount : 0
endCount   : 20
Rows       : 3

Result
------
two
one
three
```

Note that the data contains 3 entries, including the entry you just added.

7. Add a fourth data entry:

```
gfsh>put --region=regionA --key="4" --value="four"
Result      : true
Key Class   : java.lang.String
Key         : 3
Value Class : java.lang.String
Old Value   : <NULL>
```

Note that only server2 is running. Because the data is replicated and persisted, all of the data is still available. But the new data entry is currently available only on server 2.

```
gfsh>describe region --name=regionA
..........................................................
Name            : regionA
Data Policy     : persistent replicate
Hosting Members : server2

Non-Default Attributes Shared By Hosting Members

 Type  | Name | Value
------ | ---- | -----
Region | size | 4
```

8. Stop the remaining cache server:

```
gfsh>stop server --name=server2
Stopping Cache Server running in /home/username/my_gemfire/server2 on ubuntu.local[40412] as server2...
Process ID: 4185
Log File: /home/username/my_gemfire/server2/server2.log
.....
```

Step 8. Restart the cache servers in parallel

In this step you restart the cache servers in parallel. Because the data is persisted, the data is available when the servers restart. Because the data is replicated, you must start the servers in parallel so that they can synchronize their data before starting.

1. Start server1. Because regionA is replicated and persistent, it needs data from the other server to start and waits for the server to start:

   ```
   gfsh>start server --name=server1 --server-port=40411
   Starting a GemFire Server in /home/username/my_gemfire/server1...
   ```

   Note that if you look in the server1.log file for the restarted server, you will see a log message similar to the following:

   ```
   [info 2015/01/14 09:08:13.610 PST server1 <main> tid=0x1] Region /regionA has pot
   entially stale data. It is waiting for another member to recover the latest data.
     My persistent id:

       DiskStore ID: 8e2d99a9-4725-47e6-800d-28a26e1d59b1
       Name: server1
       Location: /192.0.2.0:/home/username/my_gemfire/server1/.

     Members with potentially new data:
     [
       DiskStore ID: 2e91b003-8954-43f9-8ba9-3c5b0cdd4dfa
       Name: server2
       Location: /192.0.2.0:/home/username/my_gemfire/server2/.
     ]
     Use the gfsh "list-missing-disk-stores" command to see all disk stores that
   are being waited on by other members.
   ```

2. In a second terminal window, change directories to the scratch working directory (for example, my_gemfire) and start gfsh:

   ```
   [username@localhost ~/my_gemfire]$ gfsh
     _________________________     __
    / _____/ _______/ _______/ /____/ /
   / /  __/ /___  /_______  / _______  /
  / /__/ / ____/  _______/ / /    / /
 /_______/_/      /_______/_/    /_/

 Monitor and Manage GemFire
   ```

3. Run the following command to connect to the cluster:

   ```
   gfsh>connect --locator=localhost[10334]
   Connecting to Locator at [host=localhost, port=10334] ..
   Connecting to Manager at [host=ubuntu.local, port=1099] ..
   Successfully connected to: [host=ubuntu.local, port=1099]
   ```

4. Start server2:

```
gfsh>start server --name=server2 --server-port=40412
```

When server2 starts, note that server1 completes its start up in the first gfsh window:

```
Server in /home/username/my_gemfire/server1 on ubuntu.local[40411] as server1 is currently online.
Process ID: 3402
Uptime: 1 minute 46 seconds
GemFire Version: 10.1.0
Java Version: 11.0.17
Log File: /home/username/my_gemfire/server1/server1.log
JVM Arguments: -Dgemfire.default.locators=192.0.2.0[10334] -Dgemfire.use-cluster-configuration=true
-XX:OnOutOfMemoryError=kill -KILL %p -Dgemfire.launcher.registerSignalHandlers=true
-Djava.awt.headless=true -Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /home/username/gf10/lib/gemfire-bootstrap-10.1.0.jar
```

5. Verify that the locator and two servers are running:

```
gfsh>list members
Name      | Id                                                                | Type
--------- | ------------------------------------------------------------------| ---—---
locator-0 | 192.168.43.99(locator-0:16523:locator)<ec><v0>:42586 [Coordinator] | Locator
locator-1 | 192.168.43.99(locator-1:16524:locator)<ec><v1>:42863               | Locator
server-2  | 192.168.43.99(server-2:16525)<v2>:47137                            | Server
server-1  | 192.168.43.99(server-1:16526)<v3>:44209                            | Server
```

6. Run a query to verify that all the data you entered with the put commands is available:

```
gfsh>query --query="select * from /regionA"

Result     : true
startCount : 0
endCount   : 20
Rows       : 5

Result
------
one
two
four
Three

NEXT_STEP_NAME : END
```

7. Stop server2 with the following command:

```
gfsh>stop server --dir=server2
Stopping Cache Server running in /home/username/my_gemfire/server2 on 192.0.2.0[40412] as server2...
Process ID: 3992
Log File: /home/username/my_gemfire/server2/server2.log
....
```

8. Run a query to verify that all the data you entered with the put commands is still available:

```
gfsh>query --query="select * from /regionA"

Result     : true
startCount : 0
endCount   : 20
Rows       : 5

Result
------
one
two
four
Three

NEXT_STEP_NAME : END
```

Step 9. Shut down the system including your locators

To shut down your cluster, do the following:

1. In the current gfsh session, stop the cluster:

   ```
   gfsh>shutdown --include-locators=true
   ```

   See shutdown.

2. When prompted, type ‘Y’ to confirm the shutdown of the cluster.

   ```
   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
   Shutdown is triggered

   gfsh>
   No longer connected to ubuntu.local[1099].
   gfsh>
   ```

3. Type exit to quit the gfsh shell.

Step 10. What to do next…

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

- Continue reading the next section to learn more about the components and concepts that were just introduced.
- To get more practice using gfsh, see Tutorial—Performing Common Tasks with gfsh.
- To learn about the cluster configuration service, see Tutorial—Creating and Using a Cluster Configuration.

Add GemFire to Your Project

Java and Spring

To add GemFire or the Spring for GemFire libraries to your applications, you must access the Broadcom Customer Support portal.

Prerequisites

1. Log in to the Broadcom Customer Support Portal with your customer credentials. For more information about login requirements, see the Download Broadcom products and software article.
2. Go to the VMware Tanzu GemFire downloads page, select VMware Tanzu GemFire, click Show All Releases.
3. Find the release named Click Green Token for Repository Access and click the Token Download icon on the right. This opens the instructions on how to use the GemFire artifact repository. At the top, the Access Token is provided. Click Copy to Clipboard. You will use this Access Token as the password.

Adding Tanzu GemFire to a Maven Project

1. To add GemFire to your Java project, you must modify your project's pom.xml file. Add the following repository definition to your pom.xml file:

```
<repository>
  <id>gemfire-release-repo</id>
  <name>Pivotal GemFire Release Repository</name>
  <url>https://packages.broadcom.com/artifactory/gemfire/</url>
</repository>
```

2. Add the following dependencies to your pom.xml file.

```
<dependencies>
  <dependency>
    <groupId>com.vmware.gemfire</groupId>
    <artifactId>gemfire-core</artifactId>
    <version>10.1.0</version>
  </dependency>
  <dependency>
    <groupId>com.vmware.gemfire</groupId>
    <artifactId>gemfire-logging</artifactId>
    <version>10.1.0</version>
  </dependency>
  <!--if using continuous queries-->
  <dependency>
    <groupId>com.vmware.gemfire</groupId>
    <artifactId>gemfire-cq</artifactId>
    <version>10.1.0</version>
  </dependency>
</dependencies>
```

3. To access these artifacts, you must add an entry to your .m2/settings.xml file:

```
<settings>
  <servers>
    <server>
      <id>gemfire-release-repo</id>
      <username>EXAMPLE-USERNAME</username>
      <password>MY-PASSWORD</password>
    </server>
  </servers>
</settings>
```

Where:

- EXAMPLE-USERNAME is your support.broadcom.com user name.
- MY-PASSWORD is the Access Token you copied in step 3 in Prerequisites.

Adding Tanzu GemFire to a Gradle Project

To add Tanzu GemFire to a Gradle project:

1. Add the GemFire repository to the build.gradle file:

```
repositories {
  maven {
    credentials {
      username "gemfireRepoUsername"
      password "gemfireRepoPassword"
    }
    url = uri("https://packages.broadcom.com/artifactory/gemfire/")
  }
}
```

2. Add your credentials to the local .gradle/gradle.properties or project gradle.properties file.

```
gemfireRepoUsername=MY-USERNAME
gemfireRepoPassword=MY-PASSWORD
```

   Where:

   - EXAMPLE-USERNAME is your support.broadcom.com user name.
   - MY-PASSWORD is the Access Token you copied in step 3 in Prerequisites.

3. Add the dependencies to the project by adding the following to your build.gradle file.

```
dependencies {
  implementation "com.vmware.gemfire:gemfire-core:GEMFIRE-VERSION"
  implementation "com.vmware.gemfire:gemfire-logging:GEMFIRE-VERSION"
  // if using continuous queries
  implementation "com.vmware.gemfire:gemfire-cq:GEMFIRE-VERSION"
}
```

   Where GEMFIRE-VERSION is the version of VMware Tanzu GemFire being used for the project.

Java Client Example

For the following Java client example, start a simple cluster and create an example region as follows:

1. With Tanzu GemFire installed or available, in a terminal start the Tanzu GemFire shell (gfsh):

```
gfsh
```

2. In gfsh, run the following command to start a locator:

```
start locator
```

3. In gfsh, run the following command to start a server:

```
start server
```

4. In gfsh, run the following command to create a region named “helloWorld”:

```
create region --name=helloWorld --type=PARTITION
```

5. This example:

- Creates a ClientCache, with cluster connection information.
- Creates a client side PROXY Region (“helloWorld”) that represents the helloWorld region that you created on the server.
- PUTs data into the helloWorld region.
- GETs that data from the helloWorld region.
- Outputs to the command line.

```
import org.apache.geode.cache.Region;
import org.apache.geode.cache.client.ClientCache;
import org.apache.geode.cache.client.ClientCacheFactory;
import org.apache.geode.cache.client.ClientRegionShortcut;

public class HelloWorldApplication {

  public static void main(String[] args) {
    ClientCache cache = new ClientCacheFactory().addPoolLocator("127.0.0.1", 10334).create();
    Region<String, String>
    helloWorldRegion =
      cache.<String, String>createClientRegionFactory(ClientRegionShortcut.PROXY).create("helloWorld");

    helloWorldRegion.put("1", "HelloWorldValue");
    String value1 = helloWorldRegion.get("1");
    System.out.println(value1);
    cache.close();
  }
}
```

6. Build and run the above application. The application outputs “HelloWorldValue” to the command line.

Spring Boot for GemFire

Spring Boot for VMware Tanzu GemFire provides the convenience of Spring Boot’s convention over configuration approach by using auto-configuration with Spring Framework’s powerful abstractions and highly consistent programming model to simplify the development of GemFire applications.

To use a Spring Boot for GemFire client with Tanzu GemFire, you must add the appropriate dependencies for your application. The Spring Boot for GemFire dependencies are available from broadcom.packages.com.

Spring Boot Client Example for a Maven Project

1. Add the GemFire repository by completing steps 1 - 3 in Adding Tanzu GemFire to a Maven Project above.
2. After you have set up the repository and credentials, add the following to the pom.xml file.

   You must define both the Spring Boot for GemFire dependency and an explicit dependency on the matching version of GemFire.

```
<dependencies>
  <dependency>
    <groupId>com.vmware.gemfire</groupId>
    <artifactId>spring-boot-3.1-gemfire-10.1</artifactId>
    <version>VERSION</version>
  </dependency>
  <dependency>
    <groupId>com.vmware.gemfire</groupId>
    <artifactId>gemfire-core</artifactId>
    <version>GEMFIRE-VERSION</version>
  </dependency>
  <dependency>
    <groupId>com.vmware.gemfire</groupId>
    <artifactId>gemfire-logging</artifactId>
    <version>GEMFIRE-VERSION</version>
  </dependency>
  <!--if using continuous queries-->
  <dependency>
    <groupId>com.vmware.gemfire</groupId>
    <artifactId>gemfire-cq</artifactId>
    <version>GEMFIRE-VERSION</version>
  </dependency>
</dependencies>
```

Where:

- spring-boot-3.1-gemfire-10.1 is the version of Spring Boot + the version of GemFire your application requires.
- VERSION is the current version of Spring Boot for GemFire available for the artifact version.
- GEMFIRE-VERSION is the version of GemFire being used for the project

3. Your application is now ready to connect with your GemFire instance.

Spring Boot Client Example for a Gradle Project

You must add the GemFire repository to your Spring Session for Tanzu GemFire projects. To add Spring Boot for Spring Session for Tanzu GemFire to a Gradle project:

1. Add the GemFire repository by completing steps 1 - 3 in Adding Tanzu GemFire to a Gradle Project above.
2. After you have set up the repository and credentials, add the following to the build.gradle file. You must define both the Spring Boot for GemFire dependency and an explicit dependency on the matching version of GemFire.

```
dependencies {
  implementation "com.vmware.gemfire:spring-boot-3.1-gemfire-10.1:VERSION"
  implementation "com.vmware.gemfire:gemfire-core:GEMFIRE-VERSION"
  implementation "com.vmware.gemfire:gemfire-logging:GEMFIRE-VERSION"
  // if using continuous queries
  implementation "com.vmware.gemfire:gemfire-cq:GEMFIRE-VERSION"
}
```

Where:

- spring-boot-3.1-gemfire-10.1` is the version of Spring Boot + the version of GemFire your application requires.

- VERSION is the current version of Spring Boot for GemFire available for the artifact version.
- GEMFIRE-VERSION is the version of GemFire being used for the project.

3. Your application is now ready to connect with your GemFire instance.

Confirm Dependency Usage

To confirm that your project is using the Spring for GemFire dependencies correctly, download the Hello World! application, update the appropriate files for your credentials, and run the application.

Additional Resources

- Spring Boot for Tanzu GemFire Documentation
- Spring Data for Tanzu GemFire Documentation
- Spring Session for Tanzu GemFire Documentation
- Spring Integration for Tanzu GemFire Documentation
- Spring Cloud Data Flow for Tanzu GemFire Documentation

.NET Framework Native Client

To use the VMware Tanzu GemFire .NET Framework Native Client, download the VMware Tanzu GemFire .NET Framework Client library from the Broadcom Customer Support Portal.

1. Log in to the Broadcom Customer Support Portal with your customer credentials. Before downloading, ensure you meet the requirements, for more information, see the Download Broadcom products and software article.
2. Go to the VMware Tanzu GemFire downloads page, and select VMware Tanzu GemFire.
3. Select Native Client 10.4.3.
4. Review the Terms and Conditions, and then click I agree to Terms and Conditions.
5. Click the HTTPS Download icon for the Native Client file your environment.
6. Decompress the distribution archive, which might be a ZIP archive or a compressed tar file (.tar.gz or .tgz). For example:

   ```
   unzip pivotal-gemfire-nativeclient-windows-64bit-10.x.y.zip
   ```

   or

   ```
   tar xvzf pivotal-gemfire-nativeclient-linux-64bit-10.x.y.tar.gz
   ```

7. Add the VMware.GemFire.dll and VMware.GemFire.pdb files to your project.
8. In the .csproj file, add a reference to the VMware.GemFire.dll library.

   ```
   <ItemGroup>
     <Reference Include="VMware.GemFire.dll">
       <HintPath>..\VMware.GemFire.dll</HintPath>
     </Reference>
   </ItemGroup>
   ```

Example: Put, Get, and Remove with VMware Tanzu GemFire Native .NET Client (C#)

```
using System;
using Apache.Geode.Client;


 class Program
 {
   static void Main(string[] args)
   {
     var cache = new CacheFactory()
       .Set("log-level", "none")
       .Create();

     cache.GetPoolManager()
       .CreateFactory()
       .AddLocator("localhost", 10334)
       .Create("pool");

     var regionFactory = cache.CreateRegionFactory(RegionShortcut.PROXY)
       .SetPoolName("pool");
     var region = regionFactory.Create<string, string>("example_userinfo");

     Console.WriteLine("Storing id and username in the region");

     const string rtimmonsKey = "rtimmons";
     const string rtimmonsValue = "Robert Timmons";
     const string scharlesKey = "scharles";
     const string scharlesValue = "Sylvia Charles";

     region.Put(rtimmonsKey, rtimmonsValue);
     region.Put(scharlesKey, scharlesValue);

     Console.WriteLine("Getting the user info from the region");
     var user1 = region.Get(rtimmonsKey, null);
     var user2 = region.Get(scharlesKey, null);

     Console.WriteLine(rtimmonsKey + " = " + user1);
     Console.WriteLine(scharlesKey + " = " + user2);

     Console.WriteLine("Removing " + rtimmonsKey + " info from the region");

     if (region.Remove(rtimmonsKey))
     {
       Console.WriteLine("Info for " + rtimmonsKey + " has been deleted");
     }
     else
     {
       Console.WriteLine("Info for " + rtimmonsKey + " has not been deleted");
     }

     cache.Close();
   }
 }
```

Additional Resources

- native client for VMware Tanzu GemFire documentation

C++ Native Client

To use the VMware Tanzu GemFire C++ Native Client, download the VMware Tanzu GemFire C++ Client library from the Broadcom Customer Support Portal.

1. Log in to the Broadcom Customer Support Portal with your customer credentials. Before downloading, ensure you meet the requirements, for more information, see the Download Broadcom products and software article.
2. Go to the VMware Tanzu GemFire downloads page, and select VMware Tanzu GemFire.
3. Select Native Client 10.4.3.
4. Review the Terms and Conditions, and then click I agree to Terms and Conditions.
5. Click the HTTPS Download icon for the Native Client file your environment.
6. Decompress the distribution archive, which might be a ZIP archive or a compressed tar file (.tar.gz or .tgz). For example:

   ```
   unzip pivotal-gemfire-nativeclient-windows-64bit-10.x.y.zip
   ```

   or

   ```
   tar xvzf pivotal-gemfire-nativeclient-linux-64bit-10.x.y.tar.gz
   ```

Example: Put, Get, and Remove with VMware Tanzu GemFire Native C++ Client

```
#include <iostream>

#include <geode/CacheFactory.hpp>
#include <geode/PoolManager.hpp>
#include <geode/RegionFactory.hpp>
#include <geode/RegionShortcut.hpp>

using namespace vmware::gemfire::client;

int main(int argc, char** argv) {
  auto cache = CacheFactory()
    .set("log-level", "none")
    .create();

  cache.getPoolManager()
    .createFactory()
    .addLocator("localhost", 10334)
    .create("pool");

  auto regionFactory = cache.createRegionFactory(RegionShortcut::PROXY);
  auto region = regionFactory.setPoolName("pool").create("example_userinfo");

  std::cout << "Storing id and username in the region" << std::endl;
  region->put("rtimmons", "Robert Timmons");
  region->put("scharles", "Sylvia Charles");

  std::cout << "Getting the user info from the region" << std::endl;
  auto user1 = region->get("rtimmons");
  auto user2 = region->get("scharles");
  std::cout << "  rtimmons = "
            << std::dynamic_pointer_cast<CacheableString>(user1)->value()
            << std::endl;
  std::cout << "  scharles = "
            << std::dynamic_pointer_cast<CacheableString>(user2)->value()
            << std::endl;

  std::cout << "Removing rtimmons info from the region" << std::endl;
  region->remove("rtimmons");

  if (region->existsValue("rtimmons")) {
    std::cout << "rtimmons's info not deleted" << std::endl;
  } else {
    std::cout << "rtimmons's info successfully deleted" << std::endl;
  }

  cache.close();
}
```

Additional Resources

- native client for Tanzu GemFire documentation

Configuring and Running a Cluster

This topic links to documentation that explains how to configure and run VMware Tanzu GemFire clusters.

Use the gfsh command-line utility to configure your Tanzu GemFire cluster. The cluster configuration service persists the cluster configurations and distributes the configurations to members of the cluster. There are also several additional ways to configure a cluster.

Use gfsh to configure regions, disk stores, members, and other Tanzu GemFire objects. You can also use gfsh to start and stop locators, servers, and Tanzu GemFire monitoring tools. As you execute these commands, the cluster configuration service persists the configuration. When new members join the cluster, the service distributes the configuration to the new members.

gfsh is the recommended means of configuring and managing your VMware Tanzu GemFire cluster, however you can still configure many aspects of a cluster using the older methods of the cache.xml and gemfire.properties files. See cache.xml and the Reference for configuration parameters. You can also configure some aspects of a cluster using a Java API. See Managing VMware Tanzu GemFire.

- Overview of the Cluster Configuration Service

  The VMware Tanzu GemFire cluster configuration service persists cluster configurations created by gfsh commands to the locators in a cluster and distributes the configurations to members of the cluster.

- Tutorial—Creating and Using a Cluster Configuration

  A short walk-through that uses a single computer to demonstrate how to use gfsh to create a cluster configuration for a Tanzu GemFire cluster.

- Deploying Application JARs to VMware Tanzu GemFire Members

  You can dynamically deploy your application JAR files to specific members or to all members in your cluster.

- Using Member Groups

  VMware Tanzu GemFire allows you to organize your cluster members into logical member groups.

- Exporting and Importing Cluster Configurations

  The cluster configuration service exports and imports configurations created using gfsh for an entire VMware Tanzu GemFire cluster.

- Cluster Configuration Files and Troubleshooting

  When you use the cluster configuration service in Tanzu GemFire, you can examine the generated configuration files in the cluster_config directory on the locator. gfsh saves configuration files at the cluster-level and at the individual group-level.

- Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS

  You can connect gfsh via HTTP or HTTPS to a remote cluster and manage the cluster using gfsh commands.

- Deploying Configuration Files without the Cluster Configuration Service

You can deploy your VMware Tanzu 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.

- Starting Up and Shutting Down Your System

  Determine the proper startup and shutdown procedures, and write your startup and shutdown scripts.

- Running Tanzu GemFire Locator Processes

  The locator is a Tanzu GemFire process that tells new, connecting members where running members are located and provides load balancing for server use.

- Running Tanzu GemFire Server Processes

  A Tanzu GemFire server is a process that runs as a long-lived, configurable member of a client/server system.

- Managing System Output Files

  Tanzu GemFire output files are optional and can become excessively large. Work with your system administrator to determine where to place them to avoid interfering with other system activities.

- Firewall Considerations

  You can configure and limit port usage for situations that involve firewalls, for example, between client-server or server-server connections.

Overview of the Cluster Configuration Service

This topic provides an overview of the VMware Tanzu GemFire cluster configuration service.

The Tanzu GemFire cluster configuration service persists cluster configurations created by gfsh commands to the locators in a cluster and distributes the configurations to members of the cluster.

Why Use the Cluster Configuration Service

We highly recommend that you use the gfsh command line and the cluster configuration service as the primary mechanism to manage your cluster configuration. Specify configuration within a cache.xml file for only those items that cannot be specified or altered using gfsh. Using a common cluster configuration reduces the amount of time you spend configuring individual members and enforces consistent configurations when bringing up new members in your cluster. You no longer need to reconfigure each new member that you add to the cluster. You no longer need to worry about validating your cache.xml file. It also becomes easier to propagate configuration changes across your cluster and deploy your configuration changes to different environments.

You can use the cluster configuration service to:

- Save the configuration for an entire VMware Tanzu GemFire cluster.
- Restart members using a previously-saved configuration.
- Export a configuration from a development environment and migrate that configuration to create a testing or production system.
- Start additional servers without having to configure each server separately.
- Configure some servers to host certain regions and other servers to host different regions, and configure all servers to host a set of common regions.

Using the Cluster Configuration Service

To use the cluster configuration service in Tanzu GemFire, you must use dedicated, standalone locators in your deployment. You cannot use the cluster configuration service with co-located locators (locators running in another process such as a server).

The standalone locators distribute configuration to all locators in a cluster. Every locator in the cluster with --enable-cluster-configuration set to true keeps a record of all cluster-level and group-level configuration settings.

> The default behavior for gfsh is to create and save cluster configurations. You can deactivate the cluster configuration service by using the --enable-cluster-configuration=false option when starting locators.

You can load existing configuration into the cluster by using the gfsh import cluster-configuration command after starting up a locator.

Subsequently, any servers that you start with gfsh that have --use-cluster-configuration set to true will pick up the cluster configuration from the locator as well as any appropriate group-level configurations (for member groups they belong to). To deactivate the cluster configuration service on a server, you must start the server with the --use-cluster-configuration parameter set to false. By default, the parameter is set to true.

How the Cluster Configuration Service Works

When you use gfsh commands to create VMware Tanzu GemFire regions, disk-stores, and other objects, the cluster configuration service saves the configurations on each locator in the cluster. If you specify a group when issuing these commands, a separate configuration is saved containing only configurations that apply to the group.

When you use gfsh to start new VMware Tanzu GemFire servers, the locator distributes the persisted configurations to the new server. If you specify a group when starting the server, the server receives the group-level configuration in addition to the cluster-level configuration. Group-level configurations are applied after cluster-wide configurations; therefore you can use group-level to override cluster-level settings.

Steps that occur when you start new GemFire servers as described below.

1. Developer or Administrator executes gfsh commands to configure the cluster.
2. gfsh saves the cluster information about the locators. Existing servers using cluster configuration are configured.
3. Developer or Administrator executes gfsh to add new members to the cluster.
4. New members request the configuration from a locator.
5. Locator distributes the configuration to new servers joining the cluster. Configuration includes regions, disk stores, and jar files,

gfsh Commands that Create Cluster Configurations

The following gfsh commands cause the configuration to be written to all locators in the cluster (the locators write the configuration to disk):

- configure pdx*

- create region
- alter region**
- alter runtime
- destroy region
- create index
- destroy index
- create disk-store
- destroy disk-store
- create async-event-queue
- alter async-event-queue
- destroy async-event-queue
- deploy jar
- undeploy jar
- create gateway-sender
- destroy gateway-sender
- create gateway-receiver
- destroy gateway-receiver
- alter query-service

> The configure pdx command must be executed *before* starting your data members. This command does not affect any currently running members in the system. Data members (with cluster configuration enabled) that are started after running this command will pick up the new PDX configuration.

If cluster configuration is enabled, the region this command is applied to must exist in the cluster configuration. If that is not the case, the command will fail saying the region does not exist.

gfsh Limitations

These are the configurations that you cannot create or alter using gfsh. These configurations must be within a cache.xml file or be applied by using the API:

- Client cache configuration
- You cannot directly modify the attributes of the following objects:
    - function
    - custom-load-probe
    - compressor
    - serializer
    - instantiator
    - pdx-serializer

> The configure pdx command always specifies the org.apache.geode.pdx.ReflectionBasedAutoSerializer class. You cannot specify a custom PDX serializer in gfsh.

  - initializer
  - lru-heap-percentage
  - lru-memory-size
  - partition-resolver
  - partition-listener
  - transaction-listener
  - transaction-writer
- Adding or removing a TransactionListener
- Configuring a GatewayConflictResolver
- You cannot specify parameters and values for Java classes for the following:
  - gateway-listener
  - gateway-conflict-resolver
  - gateway-event-filter
  - gateway-event-substitution-filter

Deactivating the Cluster Configuration Service

If you do not want to use the cluster configuration service, start up your locator with the --enable-cluster-configuration parameter set to false or do not use standalone locators. You will then need to configure the cache (via cache.xml or API) separately on all your cluster members.

Tutorial—Creating and Using a Cluster Configuration

This topic provides a short walk-through that uses a single computer to demonstrate how to use gfsh to create a cluster configuration for a VMware Tanzu GemFire cluster.

The gfsh command-line tool allows you to configure and start a Tanzu GemFire cluster. The cluster configuration service uses VMware Tanzu GemFire locators to store the configuration at the group and cluster levels and serves these configurations to new members as they are started. The locators store the configurations in a hidden region that is available to all locators and also write the configuration data to disk as XML files. Configuration data is updated as gfsh commands are executed.

This section provides a walk-through example of configuring a simple VMware Tanzu GemFire cluster and then re-using that configuration in a new context.

1. Create a working directory (For example:/home/username/my_gemfire) and switch to the new directory. This directory will contain the configurations for your cluster.
2. Start the gfsh command-line tool. For example:

   ```
   $ gfsh
   ```

   The gfsh command prompt displays.

```
  _________________________    __
 / _____/ _______/ _______/ /____/ /
/ /  __/ /___  /______  / ______  /
/ /__/ / ____/  ______/ / /    / /
/_______/_/      /_______/_/    /_/    10.2

Monitor and Manage VMware Tanzu GemFire
gfsh>
```

3. Start a locator using the command in the following example:

```
gfsh>start locator --name=locator1
Starting a Tanzu GemFire Locator in /Users/username/my_gemfire/locator1...
.............................
Locator in /Users/username/my_gemfire/locator1 on 192.0.2.0[10334] as locator1
  is currently online.
Process ID: 70919
Uptime: 12 seconds
Tanzu GemFire Version: 10.2
Java Version: 1..0_361
Log File: /Users/username/my_gemfire/locator1/locator1.log
JVM Arguments: -Dgemfire.enable-cluster-configuration=true
-Dgemfire.load-cluster-configuration-from-dir=false
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /Users/username/vmware-gemfire-10.2.0/lib/gemfire-dependencies.jar

Successfully connected to: JMX Manager [host=192.0.2.0, port=1099]

Cluster configuration service is up and running.
```

Note that gfsh responds with a message indicating that the cluster configuration service is up and running. If you see a message indicating a problem, review the locator log file for possible errors. The path to the log file is displayed in the output from gfsh.

4. Start VMware Tanzu GemFire servers using the commands in the following example:

```
gfsh>start server --name=server1 --groups=group1
Starting a Tanzu GemFire Server in /Users/username/my_gemfire/server1...
.....
Server in /Users/username/my_gemfire/server1 on 192.0.2.0[40404] as server1
  is currently online.
Process ID: 5627
Uptime: 2 seconds
Tanzu GemFire Version: 10.2
Java Version: 1..0_361
Log File: /Users/username/my_gemfire/server1/server1.log

JVM Arguments: -Dgemfire.default.locators=192.0.2.0[10334] -Dgemfire.groups=group1
-Dgemfire.start-dev-rest-api=false -Dgemfire.use-cluster-configuration=true
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
Class-Path: /Users/username/vmware-gemfire-10.2.0/lib/gemfire-dependencies.jar

gfsh>start server --name=server2 --groups=group1 --server-port=40405
Starting a Tanzu GemFire Server in /Users/username/my_gemfire/server2...
.....
Server in /Users/username/my_gemfire/server2 on 192.0.2.0[40405] as server2
  is currently online.
Process ID: 5634
Uptime: 2 seconds
Tanzu GemFire Version: 10.2
Java Version: 1..0_361
Log File: /Users/username/my_gemfire/server2/server2.log

JVM Arguments: -Dgemfire.default.locators=192.0.2.0[10334] -Dgemfire.groups=group1
-Dgemfire.start-dev-rest-api=false -Dgemfire.use-cluster-configuration=true
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /Users/username/vmware-gemfire-10.2.0/lib/gemfire-dependencies.jar

gfsh>start server --name=server3 --server-port=40406
Starting a Tanzu GemFire Server in /Users/username/my_gemfire/server3...
.....
Server in /Users/username/my_gemfire/server3 on 192.0.2.0[40406] as server3
  is currently online.
Process ID: 5637
Uptime: 2 seconds
Tanzu GemFire Version: 10.2
Java Version: 1..0_361
Log File: /Users/username/my_gemfire/server3/server3.log
JVM Arguments: -Dgemfire.default.locators=192.0.2.0[10334]
-Dgemfire.start-dev-rest-api=false -Dgemfire.use-cluster-configuration=true
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /Users/username/vmware-gemfire-10.2.0/lib/gemfire-dependencies.jar
```

Note that the gfsh commands you used to start server1 and server2 specify a group named group1 while the command for server3 did not specify a group name.

5. Create some regions using the commands in the following example:

```
gfsh>create region --name=region1 --groups=group1 --type=REPLICATE
Member  | Status | Message
------- | ------ | ---------------------------------------
server1 | OK     | Region "/region1" created on "server1"
server2 | OK     | Region "/region1" created on "server2"

Cluster configuration for group 'group1' is updated.

gfsh>create region --name=region2 --type=REPLICATE
Member  | Status | Message
------- | ------ | ---------------------------------------
server1 | OK     | Region "/region2" created on "server1"
server2 | OK     | Region "/region2" created on "server2"
server3 | OK     | Region "/region2" created on "server3"

Cluster configuration for group 'cluster' is updated.
```

Note that region1 is created on all cache servers that specified the group named group1 when starting the cache server (server1 and server2, in this example). region2 is created on all members because no group was specified.

6. Deploy jar files. Use the gfsh deploy command to deploy application jar files to all members or to a specified group of members. The following example deploys the mx4j-3.0.2.jar and ra.jar files from the distribution. (Note: This is only an example, you do not need to deploy these files to use the Cluster Configuration Service. Alternately, you can use any two jar files for this demonstration.)

```
gfsh>deploy --groups=group1 --jars=/lib/mx4j-3.0.2.jar

Deploying files: mx4j-3.0.2.jar
Total file size is: 0.39MB

Continue?  (Y/n): y
Member  |  Deployed JAR  | Deployed JAR Location
------- | -------------- | ------------------------------------------------------
server1 | mx4j-3.0.2.jar | /Users/username/my_gemfire/server1/mx4j-3.0.2.v1.jar
server2 | mx4j-3.0.2.jar | /Users/username/my_gemfire/server2/mx4j-3.0.2.v1.jar

gfsh>deploy --jars=/lib/ra.jar

Deploying files: ra.jar
Total file size is: 0.03MB

Continue?  (Y/n): y
Member  | Deployed JAR | Deployed JAR Location
------- | ------------ | ------------------------------------------------
server1 | ra.jar       | /Users/username/my_gemfire/server1/ra.v1.jar
server2 | ra.jar       | /Users/username/my_gemfire/server2/ra.v1.jar
server3 | ra.jar       | /Users/username/my_gemfire/server2/ra.v1.jar
```

Note that the mx4j-3.0.2.jar file was deployed only to the members of group1 and the ra.jar was deployed to all members.

7. Export the cluster configuration. You can use the gfsh export cluster-configuration command to create a zip file that contains the cluster's persisted configuration. The zip file contains a copy of the contents of the cluster_config directory. For example:

```
gfsh>export cluster-configuration --zip-file-name=/Users/username/myClConfig.zip
```

VMware Tanzu GemFire writes the cluster configuration to the specified zip file.

```
File saved to /Users/username/myClConfig.zip
```

The remaining steps demonstrate how to use the cluster configuration you just created.

8. Shut down the cluster using the following commands:

```
gfsh>shutdown --include-locators=true
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
Shutdown is triggered

gfsh>
No longer connected to 192.0.2.0[1099].
gfsh>
```

9. Exit the gfsh command shell:

```
gfsh>quit
Exiting...
```

10. Create a new working directory (for example: new_gemfire) and switch to the new directory.
11. Start the gfsh command shell:

```
$ gfsh
```

12. Start a new locator. For example:

```
gfsh>start locator --name=locator2 --port=10335
Starting a Tanzu GemFire Locator in /Users/username/new_gemfire/locator2...
.............................
Locator in /Users/username/new_gemfire/locator2 on 192.0.2.0[10335] as locator2
  is currently online.
Process ID: 5749
Uptime: 15 seconds
Tanzu GemFire Version: 10.2
Java Version: 1..0_361
Log File: /Users/username/new_gemfire/locator2/locator2.log

JVM Arguments: -Dgemfire.enable-cluster-configuration=true
-Dgemfire.load-cluster-configuration-from-dir=false
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /Users/username/vmware-gemfire-10.2.0/lib/gemfire-dependencies.jar

Successfully connected to: JMX Manager [host=192.0.2.0, port=1099]

Cluster configuration service is up and running.
```

13. Import the cluster configuration using the import cluster-configuration command. For example:

```
gfsh>import cluster-configuration --zip-file-name=/Users/username/myClConfig.zip
This command will replace the existing cluster configuration, if any, The old configuration will be backed up in the worki
ng directory.

Continue?  (Y/n): y
Cluster configuration successfully imported
```

Note that the locator2 directory now contains a cluster_config subdirectory.

14. Start a server that does not reference a group:

```
gfsh>start server --name=server4 --server-port=40414
Starting a Tanzu GemFire Server in /Users/username/new_gemfire/server4...
........
Server in /Users/username/new_gemfire/server4 on 192.0.2.0[40414] as server4
is currently online.
Process ID: 5813
Uptime: 4 seconds
Tanzu GemFire Version: 10.2
Java Version: 1..0_361
Log File: /Users/username/new_gemfire/server4/server4.log

JVM Arguments: -Dgemfire.default.locators=192.0.2.0[10335]
-Dgemfire.start-dev-rest-api=false -Dgemfire.use-cluster-configuration=true
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /Users/username/vmware-gemfire-10.2.0/lib/gemfire-dependencies.jar
```

15. Start another server that references group1:

```
gfsh>start server --name=server5 --groups=group1 --server-port=40415
Starting a Tanzu GemFire Server in /Users/username/new_gemfire/server5...
.....
Server in /Users/username/new_gemfire/server2 on 192.0.2.0[40415] as server5
is currently online.
Process ID: 5954
Uptime: 2 seconds
Tanzu GemFire Version: 10.2
Java Version: 1..0_361
Log File: /Users/username/new_gemfire/server5/server5.log
JVM Arguments: -Dgemfire.default.locators=192.0.2.0[10335] -Dgemfire.groups=group1
-Dgemfire.start-dev-rest-api=false -Dgemfire.use-cluster-configuration=true
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /Users/username/vmware-gemfire-10.2.0/lib/gemfire-dependencies.jar
```

16. Use the list regions command to display the configured regions. Note that region1 and region2, which were configured in the original cluster level are available.

```
gfsh>list regions
List of regions
---------------
region1
region2
```

17. Use the describe region command to see which members host each region. Note that region1 is hosted only by server5 because server5 was started using the group1 configuration. region2 is hosted on both server4 and server5 because region2 was created without a group specified.

```
gfsh>describe region --name=region1
Name           : region1
Data Policy    : replicate
Hosting Members : server5

Non-Default Attributes Shared By Hosting Members

 Type  |    Name     | Value
------ | ----------- | ---------------
Region | data-policy | REPLICATE
       | size        | 0
       | scope       | distributed-ack

gfsh>describe region --name=region2
.......................................................
Name           : region2
Data Policy    : replicate
Hosting Members : server5
                  server4

Non-Default Attributes Shared By Hosting Members

 Type  |    Name     | Value
------ | ----------- | ---------------
Region | data-policy | REPLICATE
       | size        | 0
       | scope       | distributed-ack
```

This new cluster uses the same configuration as the original system. You can start any number of servers using this cluster configuration. All servers will receive the cluster-level configuration. Servers that specify group1 also receive the group1 configuration.

18. Shut down your cluster using the following commands:

```
gfsh>shutdown --include-locators=true
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
Shutdown is triggered

gfsh>
No longer connected to 192.0.2.0[1099].
```

Deploying Application JARs to Tanzu GemFire Members

This topic explains how to deploy application JAR files to VMware Tanzu GemFire cluster members.

You can dynamically deploy your application JAR files to specific members or to all members in your cluster. Tanzu GemFire automatically keeps track of JAR file versions, auto-loads the deployed JAR files to the class loader, and auto-registers any functions that the JAR contains.

To deploy and undeploy application JAR files in VMware Tanzu GemFire, use the gfsh deploy or undeploy command.

You can deploy a single JAR or multiple JARs (by specifying either the JAR filenames or a directory that contains the JAR files), and you can also target the deployment to a member group or multiple member groups. For example, after connecting to the cluster where you want to deploy the JAR files, you could type at the gfsh prompt:

```
gfsh> deploy --jars=group1_functions.jar
```

This command deploys the group1_functions.jar file to all members in the cluster.

To deploy the JAR file to a subset of members, use the --groups argument. For example:

```
gfsh> deploy --jars=group1_functions.jar --groups=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 Configuring and Running a Cluster for more information about defining member groups and adding 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 cluster:

```
gfsh> undeploy
```

To undeploy a specific JAR file:

```
gfsh> undeploy --jars=group1_functions.jar
```

To target a specific member group when undeploying all JAR files:

```
gfsh> undeploy --groups=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 JAR files in your cluster:

```
gfsh> list deployed
```

To see a list of all deployed JARs in a specific member group:

```
gfsh> list deployed --groups=MemberGroup1
```

Sample output:

```
 Member   |     Deployed JAR     |                JAR Location
--------- | -------------------- | ------------------------------------------------
datanode1 | group1_functions.jar | /usr/local/gemfire/deploy/group1_functions.v1.jar
datanode2 | group1_functions.jar | /usr/local/gemfire/deploy/group1_functions.v1.jar
```

For more information about gfsh usage, see gfsh.

About Deploying JAR Files and the Cluster Configuration Service

By default, the cluster configuration service distributes deployed JAR files to all locators in the cluster. When you start a new server using gfsh, the locator supplies configuration files and deployed jar files to the member and writes them to the server's directory.

See Overview of the Cluster Configuration Service.

Tanzu GemFire deploys JAR files using a classloader isolation model. JAR files are loaded into their own classloaders which are isolated from the rest of the system and from each other. Deployments can access classes from other deployed JAR files and from the system, but will look inside their own classloaders first before looking externally.

> Classloader isolation is the default setting and is recommended for all new implementations. Chained classloading was the GemFire version 9 default, and is provided in support of legacy applications. Classloader isolation can be disabled using the flag –disable-classloader-isolation=true when starting members through Gfsh using the start command.

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.

Naming JAR Files

- When deploying a JAR, the name of the JAR cannot contain any of the following characters: *, !, #, @, ?.
- Use a naming format similar to the following:

```
{artifactName}-{version}.jar
```

Where

artifactName is the artifact, for example, domainModels
version should follow the SemVer Major.Minor.Patch format

Deployment Location for JAR Files

JAR files are placed into a staging directory on the system and are not copied to the server's working directory. Since JAR files are not located in the working directory, they will not be automatically redeployed from disk when the server restarts. JAR files will be redeployed by cluster configuration if it has been enabled.

Versioning of JAR Files

When you deploy a new JAR file, the member receiving the deployment checks whether the JAR file is a duplicate, meaning a JAR file with the same name and matching byte-for-byte contents is already deployed. If the JAR file is determined to be a duplicate, the deployment succeeds, but no changes occur as a result. If the JAR file is not a duplicate, the old JAR file is removed, its classes are unloaded, and the new JAR file is deployed.

Automatic Class Path Loading

Once a JAR file is deployed, its classes and resources are loaded into a classloader which will be accessible via the ClassPathLoader.

Undeploying a JAR file automatically unloads the classes and functions that were loaded during deployment. You do NOT need to restart your members to unload those classes.

JAR files are not automatically loaded from disk when the member starts. However, JAR files are automatically deployed by cluster configuration if it is enabled.

Classloader Isolation

This topic explains Classloader Isolation in VMware Tanzu GemFire.

Classloader Isolation is enabled by default.

Members started through gfsh are automatically started with classloader isolation. You can disable classloader by using the --disable-classloader-isolation option on the start command. See Disable Classloader Isolation.

With classloader isolation, Tanzu GemFire's classes are loaded into one module and user deployed jars and extensions are each deployed into their own modules. Additional class path entries provided to the start command by the --classpath option are also loaded into a separate module. Each module contains its own classloader.

Modules can access classes from other classloaders, but will first attempt to resolve classes internally, avoiding conflicts with other modules which may contain different versions of the same classes.

Loaded classloader modules and the relationships between them are logged out at member startup and when extensions or modules are loaded or unloaded.

In most circumstances, there should be no change in behavior when using classloader isolation, but there are differences:

- Deployed jars will not be loaded from disk when restarting a member when using classloader isolation. Jars will still be deployed from cluster configuration if enabled.
- The --include-system-classpath option is not valid when using classloader isolation.
- Deployed jars are not versioned when deploying updates.
- Deployments can use different versions of the same libraries without conflicting with each other.

Disable Classloader Isolation

In some situations or for diagnostic purposes, you may need to start GemFire processes without support for classloader isolation and extensions. For each process for which you want to disable classloader isolation, run the corresponding command from the sections below. These commands set flags for the processes.

Setting one of these flags prevents the process from loading any GemFire extension, like GemFire Search or GemFire for Redis Apps. Additionally, classloader isolation is removed from the process and all libraries are added to the system class path, similar to previous releases of GemFire.

gfsh

To start gfsh without classloader isolation and extensions, run:

```
DISABLE_CLASSLOADER_ISOLATION=true /path/to/gfsh
```

Locators

To start locators without classloader isolation and extensions, run:

```
gfsh>start locator ... --disable-classloader-isolation
```

Servers

To start servers without classloader isolation and extensions,run:

```
gfsh>start server ... --disable-classloader-isolation
```

Deploying Application JARs Using the Chained Approach

This topic provides implementation-specific information for deploying JAR files when not using Classloader Isolation. Chained class loading was the default in Tanzu GemFire version 9.x, and is provided for legacy support.

Classloader isolation can be disabled using the flag --disable-classloader-isolation=true when starting servers through gfsh using the start server command. With the non-classloader isolated, or chained, approach, JAR files are loaded into classloaders which are chained together with the system classloader where they can access other classes in the hierarchy.

For information common between all deployment implementation, see Deploying Application Jars.

Deployment Location for JAR Files (chained model)

The system location where JAR files are written on each member is determined by the deploy-working-dir Tanzu 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 class loader and auto-registers functions.

Versioning of JAR Files (chained model)

When you deploy JAR files to a cluster or member group, the name of the JAR file is modified to indicate version information. Each JAR filename contains a version number inserted just before the .jar suffix. For

example, if you deploy MyClasses.jar five times, the filename is displayed as MyClasses.v5.jar when you list all deployed JAR files.

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 (chained model)

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 class loader 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.

Using Member Groups

VMware Tanzu GemFire allows you to organize your cluster members into logical member groups.

The use of member groups in Tanzu 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 in gfsh.
- Perform certain disk operations like disk-store compaction across a member group. See Disk Store Commands 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 or upon member startup in gfsh.

> Any roles defined in the currently existing roles property will now be considered a group. If you wish to add membership roles to your cluster, 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 start up a member in gfsh and pass in the --groups 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 and Organizing Servers Into Logical Member Groups 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.

Exporting and Importing Cluster Configurations

This topic explains how to export and import cluster configurations in VMware Tanzu GemFire using the cluster configuration service.

The cluster configuration service exports and imports configurations created using gfsh for an entire Tanzu GemFire cluster.

The cluster configuration service saves the cluster configuration as you create regions, disk-stores and other objects using gfsh commands. You can export this configuration as well as any jar files that contain application files to a ZIP archive and then import this configuration to create a new cluster.

Exporting a Cluster Configuration

Issue the gfsh export cluster-configuration command to save the configuration data for your cluster in a ZIP archive. This ZIP file contains subdirectories for cluster-level configurations and a directory for each group specified in the cluster. The contents of these directories are described in Cluster Configuration Files and Troubleshooting.

To export a cluster configuration, run the gfsh export cluster-configuration command while connected to a Tanzu GemFire cluster. For example:

```
export cluster-configuration --zip-file-name=/home/username/configs/myConfig.zip
```

See export cluster-configuration.

> gfsh only saves cluster configuration values for configurations specified using gfsh. Configurations created by the management API are not saved with the cluster configurations.

Importing a Cluster Configuration

Use the gfsh import cluster-configuration command to configure a new cluster based on a configuration exported from another system. You can import a cluster configuration only into a new cluster, or into a running cluster that has not yet been configured and contains no defined regions. That is, when:

- There are no running cache servers

or

- The only running cache servers meet all of the following criteria:
    - Have been recently started
    - Have no regions defined in them
    - Have been given no other configuration changes since they started

After you have imported the configuration, any servers you start receive this cluster configuration.

To import a cluster configuration, start one or more locators and then run the gfsh import cluster-configuration command. For example:

```
import cluster-configuration --zip-file-name=/home/username/configs/myConfig.zip
```

See import cluster-configuration.

Cluster Configuration Files and Troubleshooting

This topic explains how to troubleshoot issues with VMware Tanzu GemFire cluster configuration files.

When you use the cluster configuration service in Tanzu GemFire, you can examine the generated configuration. The gfsh export cluster-configuration command outputs configured properties, the configuration on a per-group basis or for the entire cluster, and the list of deployed JAR files.

If the output is written to either a ZIP file or an XML file, you can import this configuration to a new cluster. See Exporting and Importing Cluster Configurations.

When you deploy a JAR file, the JAR file is added to a created directory called cluster_config within the locator's directory of files. Within cluster_config will be another directory named for the member group that has the configuration. For configurations that apply to all members of a cluster, the directory is named either cluster or the name specified when starting up the locator with the --cluster-config-dir option.

Troubleshooting Tips

- When you start a locator using gfsh, you should see the following message:

  ```
  Cluster configuration service is up and running.
  ```

  If you do not see this message, there may be a problem with the cluster configuration service. Use the status cluster-config-service command to check the status of the cluster configuration.

    - If the command returns RUNNING, the cluster configuration is running normally.
    - If the command returns WAITING, run the status locator command. The output of this command returns the cause of the WAITING status.

- When adding a cache.xml file for a server to the shared gfsh configuration, there is a specific order in how configuration sources are applied. When the same entity is configured more than once (such as in gfsh cluster configuration, gfsh group configuration, or a server's cache.xml file), the last configuration takes precedence. Therefore, group-level configurations take precedence over cluster-wide configurations, and cache.xml file configurations take precedence over both. However,

a combination can be used; for instance, gfsh cluster configuration can specify a setting that is not overridden by a gfsh group configuration or a server's cache.xml file. Additionally, a server can override a setting established at the cluster or group level by redefining it in its own cache.xml file.

- If a server start fails with the following exception: ClusterConfigurationNotAvailableException, the cluster configuration service may not be in the RUNNING state. Because the server requests the cluster configuration from the locator, which is not available, the start server command fails.
- You can determine what configurations a server received from a locator by examining the server's log file. See Logging.
- If a start server command specifies a cache.xml file that conflicts with the existing cluster configuration, the server startup may fail.
- If a gfsh command fails because the cluster configuration cannot be saved, the following message displays:

```
Failed to persist the configuration changes due to this command,
Revert the command to maintain consistency. Please use "status cluster-config-service"
to determine whether Cluster configuration service is RUNNING."
```

- There are some types of configurations that cannot be made using gfsh. See gfsh Limitations.

Sizing a GemFire Cluster

This topic describes Tanzu GemFire cluster sizing.

Overview

Sizing a Tanzu GemFire deployment is a process that involves calculation, as well as experimentation and testing. Some experimentation and testing is required to arrive at reasonably accurate values for the key sizing parameters that will work well in practice. This experimentation and testing involves representative data and workload, starting at a very small scale.

Experimentation and testing is required because memory overhead can vary greatly due to variations in data and workload. This makes it impractical to calculate the overhead precisely, as it is a function of too many variables, many of which stem from the Java runtime environment (JVM) and its memory management mechanism.

Resource Considerations

Memory is the primary means of data storage in Tanzu GemFire and is the first resource to consider for sizing purposes.

Horizontal scaling to satisfy memory requirements also scales out all the other hardware resources, the CPU, network, and disk. Because of this, when memory requirements are satisfied and the adequate cluster size is determined, often only small adjustments are needed to cover all the other required resources and complete the sizing process.

Typically, memory drives horizontal scaling, but it can be any of the hardware resources. In addition to hardware resources, soft resources should be considered. The most important software resources to consider are file descriptors, mostly for sockets in use, and threads (processes).

Experimentation and Testing

To size a Tanzu GemFire cluster:

1. Deploy a small representative data set and workload in a small cluster.
2. Tune the cluster to the desired performance.
3. Scale out the cluster while ensuring that key performance metrics stay within the desired SLA.

Testing at full scale is ideal, if you have sufficient resources available to use in testing. If sufficient resources are not available to use in testing, you can scale out multiple times, a few nodes at a time, to provide data points to use to project resource usage at full scale. This is an iterative process that involves analysis and tuning at each step. Tanzu GemFire statistics can assist in this analysis.

For large scale deployments that involve large data volumes, the general guideline is to scale vertically as much as possible to fit as much data as possible in a single GemFire instance. This helps minimize the size of the cluster. The limit to vertical scaling may depend on the desired SLA around node failure.

Requirements and Assumptions

To maximize the accuracy of your Tanzu GemFire cluster sizing, and to minimize unexpected situations in your production environment, VMware recommends that you first run tests to characterize memory and other resource usage under a representative workload.

Requirements:

- A subset of representative data. Typically, representative data that more closely matches the real data produces more accurate results.
- A matching subset of workload that matches the production workload as closely as possible.
- Hardware resources for testing, ideally the same category as would be used in production: the same CPU, memory, network, and disk resources per node. At a minimum, you must be able to run three GemFire data nodes to start, then be able to add a few more node to validate the scalability. In addition, you must have the same number of hosts for GemFire clients to be able to create an adequate workload.
- Familiarity with key GemFire concepts and features, such as partitioned regions, serialization, etc.

You should follow the documented best practices, such as the JVM GC configuration (CMS and ParNew), and use the currently supported platforms.

Architectural and Design Considerations

Before a sizing effort can start, the overarching architectural decisions have to be made, such as which GemFire regions to use for different types of data or what redundancy level to use. The results of sizing can inform architectural and design decisions for which multiple options are possible.

Serialization

Serialization can make a significant difference in the per-entry data overhead in memory, and subsequently in the overall memory requirements.

GemFire's PDX serialization is a serialization format that keeps data in a usable serialized form. This allows most operations on data entries without having to deserialize them, resulting in both space and performance benefits. These qualities make the PDX serialization the recommended serialization approach for most use cases.

DataSerializable is another GemFire serialization mechanism that is more space efficient than either PDX or Java Serializable. However, unlike PDX, it requires deserialization on any kind of access.

Per-entry Memory Overhead

Listed below are factors that can have significant impact on the memory overhead for data on a per entry basis, as well as performance:

- Choice of GemFire region type: Different regions have different per-entry overheads. This overhead is documented below and is included in the sizing spreadsheet.
- Choice of the serialization mechanism: Tanzu GemFire offers multiple serialization options, as well as the ability to have values stored serialized. As mentioned above, Tanzu GemFire PDX serialization is the generally recommended serialization mechanism due to its space and performance benefits.
- Choice of Keys: Smaller and simpler keys are more efficient in terms of both space and performance.
- Use of indexes: Indexing incurs a per-entry overhead, as documented in Memory Requirements for Cached Data.

If the data value objects are small but great in number, the per-entry overhead can add up to a significant memory requirement. You can reduce this overhead by grouping multiple data values into a single entry or by using containment relationships. For example, you can choose to have your Order objects contain their line items instead of having a separate OrderLineItems region. If this option is available, using it may yield performance improvements in addition to space savings.

Partitioned Region Scalability

Tanzu GemFire partitioned regions scale out by rebalancing their data buckets (partitions) to distribute the data evenly across all available nodes in a cluster. When new nodes are added to the cluster, rebalancing causes some buckets to move from the old to the new nodes such that the data is evenly balanced across all the nodes. For this to work effectively, with the end result being a well-balanced cluster, there should be at least one order of magnitude more buckets than data nodes for each partitioned region.

Typically, increasing the number of buckets improves data distribution. However, since the number of buckets cannot be changed dynamically and without downtime, the projected horizontal scale-out taken into account when determining the optimal number of buckets. Otherwise, as the system scales out over time, the data may become less evenly distributed. In the extreme case, when the number of nodes exceeds the number of buckets, adding new nodes has no effect, and the ability to scale out is lost.

Related to this is the choice of data partitioning scheme, the goal of which is to yield even data and workload distribution in the cluster. If problem with the partitioning scheme exists, the data, and likely the workload, will not be evenly balanced, and scalability will be lost.

Redundancy

Typically, choice of redundancy may be driven by data size and by whether or not data can be retrieved from some other backing store besides Tanzu GemFire. Other considerations might also be a factor in the decision.

For example, Tanzu GemFire can be deployed in an active/active configuration in two data centers such that each can take on the entire load, but will do so only if necessitated by a failure. Typically, in such deployments there are four live copies of the data at any time, with two in each datacenter. If two nodes failed in a single datacenter, the other datacenter would take over the entire workload until those two nodes were restored. This avoids data loss in the first datacenter. You could instead set redundancy to two, for a total of three copies of data). This would provide high availability even in case of a single node failure, and avoids paying the price of rebalancing when a single node fails. In this case, instead of rebalancing, a single failed node is restarted, while two copies of data still exist.

Relationship Between Horizontal and Vertical Scale

For deployments that can grow very large, you should allow for the growth by taking advantage of not just horizontal scalability, but also the ability to store as much data as possible in a single node. GemFire has been deployed in clusters of over 100 nodes. However, smaller clusters are easier to manage. So, as a general rule, you should store as much data as possible in a single node while maintaining a comfortable data movement requirement for re-establishing the redundancy SLA after a single point of failure. GemFire has been used with heaps of well over 64GB in size..

NUMA Considerations

You should understand the Non-Uniform Memory Architecture (NUMA) memory boundaries when deciding on the JVM size, and VM size in virtualized deployments.

Most modern CPUs implement this kind of architecture where memory is divided across the CPUs such that memory directly connected to the bus of each CPU has very fast access whereas memory accesses by that same CPU on the other portions of memory (directly connected to the other CPUs) can pay a significant wait-state penalty for accessing data. An example is a system that has four CPUs with eight cores each and a Non-Uniform Memory Architecture that assigns each CPU its own portion of the memory. As an example, assume that the total memory on the machine is 256GB. In this case, each NUMA node is 64GB. Growing a JVM larger than 64GB on this machine will cause wait-states to be induced when the CPUs must cross NUMA node boundaries to access memory within the heap. For this reason, you should size GemFire JVMs to fit within a single NUMA node to optimize performance.

GemFire Queues

If any GemFire queueing capabilities are used, such as for WAN distribution, client subscription, or asynchronous event listeners, you should evaluate the queues' capacity in the context of the desired SLA. For example, for how long should gateway or client subscription queues be able to keep queueing events when the connection is lost? Given that, how large should the queues be able to grow? An effective way to learn the answers to these kinds of questions is to watch the queues' growth during sizing experiments, using Tanzu GemFire statistics. For more information about this, see Step 3: Vertical Sizing below.

For WAN distribution, you should evaluate the distribution volume requirements and ensure adequate network bandwidth sizing. If sites connected by the WAN gateway may be down for extended periods of time, such as days or weeks, you must overflow the gateway queues to disk and ensure that you have sufficient disk space for those queues. If you have insufficient disk space for the queues, you may need to shut off the Gateway senders to prevent running out of space.

Sizing Process

To size a GemFire cluster:

1. Domain object sizing: Produce an entry size estimate for all the domain objects. Use this with number of entries to estimate the total memory requirements.
2. Estimating total memory and system requirements: Based on the data sizes, estimate the total memory and system requirements using the sizing spreadsheet, which takes into account GemFire region overhead. This does not account for other overhead, but provides a starting point.
3. Vertical sizing: Use the results of the previous step as the starting point in configuring a three-node cluster. Vertical sizing determines the "building block" – the sizing, configuration, and workload for a single node – by experimentation.

4. Scale-out validation: Iteratively test and adjust the single “building block” node from the previous step to verify near-linear scalability and performance.
5. Projection to full scale: Use the results of scale-out validation to arrive at the sizing configuration that meets your desired capacity and SLA.

The following sections provide details about each step.

Step 1: Domain object sizing

Before you can make any other estimates, you must estimate the size of the domain objects to be stored in the cluster.

An effective way to size a domain object is to run a single instance GemFire test with GemFire statistics enabled. In this instance, store each domain object to be sized in a dedicated partitioned region. The test loads a number of instances of each domain object, making sure they all stay in memory, with no overflow. After running the test, load the statistics file from it into VSD and examine dataStoreBytesInUse and dataStoreEntryCount partition region stats for each partitioned region. Dividing the value of dataStoreBytesInUse by the value of dataStoreEntryCount provides an estimate for the average value size that is as accurate as is possible.

> This estimate does not include the key size and entry overhead.

Another way to size domain objects is to use a heap histogram. In this approach you should run a separate test for each domain object. This simplifies the process of determining what classes are associated with data entries, based on the number of entries in memory, to figure out what classes

Step 2: Estimating total memory and system requirements

You can use the System Sizing Worksheet to approximate your total memory and system requirements. The *System Sizing Worksheet* takes into account all the GemFire region related per-entry overhead, and the desired memory headroom.

The spreadsheet formulas are rough approximations that serve to inform a high-level estimate, as they do not account for any other overhead such as buffers, threads, queues, application workload, etc. Additionally, the results obtained from the spreadsheet do not have any performance context. For this reason, the steps in Step 3: Vertical Sizing use the results for memory allocation per server obtained from the spreadsheet as the starting point for the vertical sizing process.

Step 3: Vertical Sizing

Use vertical sizing to determine what fraction of the total requirements for storage and workload can be satisfied with a single data node, and with what resources. This represents a “building block” (a unit of scale) and includes both the size of the resources and the workload capacity. It also includes the complete configuration of the building block (system, VM if present, JVM, and GemFire).

For example, a result of this step for a simple read-only application might be that a single data node with a JVM sized to 64G can store 40G of data and support a workload of 5000 get operations per second within the required latency SLA, without exhausting any resources. You should capture all the key performance indicators for the application, and verify that they meet the desired SLA. A complete output of the vertical sizing step includes all the relevant details such as hardware resources per node, peak capacity, and performance at peak capacity, and notes which resource becomes a bottleneck at peak capacity.

This approach uses experimentation to determine the optimal values for all relevant configuration settings, including the system, VM if virtualization is used, JVM, and GemFire configuration to be used.

To run experiments and tests, you musy have a cluster of three data nodes and a locator, as well as additional hosts to run clients to generate the application workload. Three data nodes are required to fully exercise the partitioning of data in partitioned regions across multiple nodes in presence of data redundancy. As a starting point, the data nodes should be sized based on the estimates obtained from the sizing spreadsheet completed in Step 2: Estimating total memory and system requirements.

Typically, the following configuration is used to begin:

- A heap headroom of 50% of the old generation
- CMSInitiatingOccupancyFraction is set to 65%
- The young generation is sized to 10% of the total heap

GemFire logging and statistics should be enabled for all the test runs. The logs should be routinely checked for problems. The statistics are analyzed for problems, verification of resources, and performance. Performance metrics can be collected by the application test logic as well. Any relevant latency metrics must be collected by the test application.

If WAN distribution is needed, you should set up an identical twin cluster and configure the WAN distribution between the two clusters. You should also size WAN capacity.

Test runs should exercise a representative application workload, with duration long enough to incur multiple GC cycles, so that stable resource usage can be confirmed. If any GemFire queues are used, run tests to determine adequate queue sizes that meet the SLA. If disk storage is used, determine adequate disk store size and configuration per node as part of this exercise.

After each test run, examine the latency metrics collected by the application. Use VSD to examine the statistics and correlate the resource usage with latencies and throughput observed. You should examine the following:

- Memory (heap, and non-heap, GC)
- CPU
- System load
- Network
- File descriptors
- Threads
- Queue statistics

For information about VSD and these statistics, see Quick Guide to Useful Statistics.

One of the objectives of vertical sizing is to determine the headroom required to accomplish the desired performance. This might take several tests to tune the headroom to no more and no less than needed. A much larger headroom than needed could amount to a significant waste of resources. A smaller headroom could cause higher GC activity and CPU usage and hurt performance.

Locator Sizing

Locator JVM sizing may be necessary when JMX Manager is running in the locator JVM, and JMX is used for monitoring. An effective way to do this is to set the locator heap to 0.5G and monitor it during the scale-out.

Notes on GC

The most important goal of GC is to avoid full GCs, as they cause pauses which can result in a GemFire data node to be unresponsive, and, as a result, be expelled from the cluster. The permanent generation space can trigger a full GC as well, which happens when it fills completely. You should size this appropriately to avoid this. Additionally, you can instruct the JVM to garbage collect the permanent generation space along with CMS GC using the following option:

```
-XX:+CMSClassUnloadingEnabled
```

You can tune GC for two of the following three:

- Latency
- Throughput
- Memory footprint

Heap headroom is important because with GemFire we sacrifice the memory footprint to accomplish latency and throughput goals.

Long minor GC pauses can be shortened by reducing the young generation. This will likely increase the frequency of minor collections. Additionally, for very large heaps, for example those of 64G and above, the old generation impact on minor GC pauses may be reduced by using the following GC settings:

```
-XX:+UnlockDiagnosticVMOptions XX:ParGCCardsPerStrideChunk=32768
```

Step 4: Scale-out Validation

During this step, you scale out the initial three node cluster at least twice, adding at least a few nodes each time. You should also scale out the client hosts accordingly to be able to create adequate workload at each step. You should increase the workload proportionally to the scale-out.

There is no definitive rule about how much to increase the cluster size, or in what increments. Typically, this determination is dictated by available hardware resources.

The goal of this step is to validate the “building block” configuration and capacity at some, larger than initial, scale. This allows you to project the capacity to full scale with confidence. You may need to tune the configuration at various points. For example, when you add more nodes to the cluster, more socket connections, buffers, and threads will be in use on each node, resulting in higher memory usage per node (both heap and non-heap), as well as increased file descriptors in use.

If you use JMX for monitoring, watch the heap usage of the locator running the JMX Manager.

Step 5: Projection to Full Scale

After you have completed Step 4: Scale-out Validation, you can determine the total cluster size. You know the storage and workload capacity of a single node and that you can scale horizontally to meet the full requirements. Additionally, you have already tuned the cluster configuration to meet the demands of the application workload.

Sizing Quick Reference

General recommendations to use as a starting point in capacity planning and sizing

Data Node Heap Size	Use
Up to 32GB	Smaller data volumes (up to a few hundred GB); very low latency required
64GB+	Larger data volumes (500GB+)

Data Node Heap Size	Use
CPU Cores per Data Node	Use
2 to 4	Development; smaller heaps
6 to 8	Production; performance/system testing; larger heaps
Network Bandwidth	Use
1GbE	Development
High bandwidth (e.g. 10GbE)	Production; performance/system testing
Disk Storage	Use
DAS, or SAN	Always
NAS	Do not use; performance and resilience issues

- Memory/CPU relationship: mind the NUMA boundary
- Virtualization: Do not oversubscribe resources (memory, CPU, storage). Run a single GemFire data node JVM per VM.

Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS

This topic explains how to use gfsh to manage a remote VMware Tanzu GemFire cluster over HTTP or HTTPS. You connect gfsh through HTTP or HTTPS to the remote cluster and manage the cluster using gfsh commands.

To connect gfsh using the HTTP protocol to a remote cluster:

1. Launch gfsh. See Starting gfsh.
2. When starting the remote cluster on the remote host, you can optionally specify --http-bind-address and --http-service-port as Tanzu GemFire properties when starting up your JMX manager (server or locator). These properties can be then used in the URL used when connecting from your local system to the HTTP service in the remote cluster. For example:

   ```
   gfsh>start server --name=server1 --J=-Dgemfire.jmx-manager=true \
   --J=-Dgemfire.jmx-manager-start=true --http-service-port=8080 \
   --http-service-bind-address=myremotecluster.example.com
   ```

   This command must be executed directly on the host machine that will ultimately act as the remote server that hosts the HTTP service for remote administration. (You cannot launch a server remotely.)
3. On your local system, run the gfsh connect command to connect to the remote system. Include the --use-http and --url parameters. For example:

   ```
   gfsh>connect --use-http=true --url="http://myremotecluster.example.com:8080/gemfire/v1"

   Successfully connected to: Tanzu GemFire Manager's HTTP service @ http://myremotecluster.example.com:8080/gemfire/v1
   ```

   See connect.

   gfsh is now connected to the remote system. Most gfsh commands will now execute on the remote system. The following commands are executed on the local cluster:

   - alter disk-store

- compact offline-disk-store
- describe offline-disk-store
- help
- hint
- sh (for executing OS commands)
- sleep
- start jconsole (however, you can connect JConsole to a remote cluster when gfsh is connected to the cluster via JMX)
- start jvisualvm
- start locator
- start server
- start vsd
- status locator *
- status server *
- stop locator *
- stop server *
- run (for executing gfsh scripts)
- validate disk-store
- version

* You can stop and obtain the status of *remote locators and servers* when gfsh is connected to the cluster via JMX or HTTP/S by using the --name option for these stop and status commands. If you use the --pid or --dir option for these commands, then the stop or status commands are executed only locally.

To configure SSL for the remote connection (HTTPS), enable SSL for the http component in gemfire.properties or gfsecurity-properties or upon server startup. See SSL for details on configuring SSL parameters. These SSL parameters also apply to all HTTP services hosted on the configured JMX Manager, which can include the following:

- Developer REST API service
- Pulse monitoring tool. (GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

Deploying Configuration Files without the Cluster Configuration Service

This topic explains how to deploying VMware Tanzu GemFire configuration files without the cluster configuration service.

You can deploy your Tanzu 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.

> If you use the cluster configuration service to create and manage your VMware Tanzu GemFire cluster configuration, the procedures described in this section are not needed

because Tanzu GemFire automatically manages the distribution of the configuration files and jar files to members of the cluster. See Overview of the Cluster Configuration Service.

You can use the procedures described in this section to distribute configurations that are member-specific, or for situations where you do not want to use the cluster configuration service.

- Main Steps to Deploying Configuration Files

  These are the basic steps for deploying configuration files, with related detail in sections that follow.

- 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.

- Changing the File Specifications

  You can change all file specifications in the gemfire.properties file and at the command line.

- Deploying Configuration Files in JAR Files

  This section provides a procedure and an example for deploying configuration files in JAR files.

Main Steps to Deploying Configuration Files

This topic describes the basic steps for deploying VMware Tanzu GemFire configuration files.

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 class path. You can specify a member's class path by using the --classpath option for the gfsh start locator and start server commands.

Tanzu GemFire Configuration Files

- gemfire.properties. Contains the settings required by members of a cluster. These settings include licensing, system member discovery, communication parameters, logging, and statistics. See the Tanzu GemFire Properties Reference.
- 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 the Tanzu GemFire Properties Reference.
- 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.

Default File Specifications and Search Locations

This topic lists VMware Tanzu GemFire 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 pathnames and filenames.

> 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. class loader	As a Java system property, use gemfirePropertyFile
cache.xml	1. current directory 2. class loader	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 Tanzu GemFire system member from /testDir in a Unix file system, Tanzu 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 class loader for test1/gfprops

Changing the File Specifications

This topic explains how to change VMware Tanzu GemFire file specifications.

You can change all file specifications in the gemfire.properties file and at the command line.

Tanzu GemFire applications can use the API to pass java.lang.System properties to the cluster connection. This changes file specifications made at the command line and in the gemfire.properties file. 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 files:

```
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>start server \
--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.

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
locators=cache-xml-file=/gemfireSamples/examples/dist/cacheRunner/queryPortfolios.xml
```

Deploying Configuration Files in JAR Files

This topic provides a procedure and an example for deploying VMware Tanzu GemFire configuration files in JAR files.

Procedure

1. Jar the files.
2. Set the Tanzu GemFire system properties to point to the files as they reside in the jar file.
3. Include the JAR file in your class path.
4. Verify the JAR file copies are the only ones visible to the application at runtime. Tanzu GemFire searches the class path 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 class path to include my.jar.
3. Verify there is no file already in the filesystem named ./myConfig/myCache.xml, so Tanzu GemFire will be forced to search the JAR file to find it.

When you start your application, the configuration file is loaded from the JAR file.

Starting Up and Shutting Down Your System

This topic explains how to start up and shut down your VMware Tanzu GemFire system.

Determine the proper startup and shutdown procedures, then write your startup and shutdown scripts.

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 Tanzu

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 gfsh.

Starting Up Your System

You should follow certain order guidelines when starting your Tanzu GemFire system.

Start servers before you start their client applications. In each cluster, follow these guidelines for member startup:

- Start locators first. See Running Tanzu GemFire Locator Processes for examples of locator start up commands.
- Start cache servers before the rest of your processes unless the implementation requires that other processes be started ahead of them. See Running Tanzu GemFire Server Processes for examples of server start up commands.
- If your cluster 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.
- For a system that includes persistent regions, see Start Up and Shut Down with Disk Stores.
- If you are running producer processes and consumer or event listener processes, start the consumers first. This ensures the consumers and listeners do not miss any notifications or updates.
- If you are starting up your locators and peer members all at once, you can use the locator-wait-time property (in seconds) upon process start up. This timeout allows peers to wait for the locators to finish starting up before attempting to join the cluster.

  If the process cannot initially reach a locator, it will sleep for join-retry-sleep milliseconds between retries until it either connects or the number of seconds specified in locator-wait-time has elapsed. By default, locator-wait-time is set to zero meaning that a process that cannot connect to a locator upon startup will throw an exception.

> You can optionally override the default timeout period for shutting down individual processes. This override setting must be specified during member startup. See Shutting Down the System for details.

Starting Up After Losing Data on Disk

This information pertains to catastrophic loss of Tanzu 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
```

> This gfsh command requires that you be connected to the cluster via a JMX Manager node.

Shutting Down the System

Shut down your Tanzu GemFire system by using either the gfsh 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 cluster 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
```

By default, the shutdown command will only shut down data nodes. If you want to shut down all nodes including locators, specify the --include-locators=true parameter. For example:

```
gfsh>shutdown --include-locators=true
```

This will shut down all locators one by one, shutting down the manager last.

To shutdown all data members after a grace period, specify a time-out option (in seconds).

```
gfsh>shutdown --time-out=60
```

To shutdown all members including locators after a grace period, specify a time-out option (in seconds).

```
gfsh>shutdown --include-locators=true --time-out=60
```

Shutting Down System Members Individually

If you are not using persistent regions, you can shut down the cluster by shutting down each member in the reverse order of their startup. (See Starting Up Your System for the recommended order of member startup.)

Shut down the cluster 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 Tanzu GemFire-connected client applications that are running in the cluster.
- Shut down any cache servers. To shut down a server, issue the following gfsh command:

  ```
  gfsh>stop server --name=<...>
  ```

  or

  ```
  gfsh>stop server --dir=<server_working_dir>
  ```

- Shut down any locators. To shut down a locator, issue the following gfsh command:

  ```
  gfsh>stop locator --name=<...>
  ```

  or

  ```
  gfsh>stop locator --dir=<locator_working_dir>
  ```

- Do not use the command line kill -9 to shut down a server under ordinary circumstances. Especially on systems with a small number of members, using a kill instead of a gfsh stop can cause the partition detection mechanism to place the system in an end state that will wait forever to reconnect to the terminated server, and there will be no way to restart that terminated server. If a kill command appears the only way to rid the system of a server, then kill *all* the processes of the cluster or use kill -INT, which will allow an orderly shutdown of the process.

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, Tanzu 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.

Running GemFire Locator Processes

---

title: Running VMware Tanzu GemFire Locator Processes

This topic explains how to run a VMware Tanzu GemFire locator process.

A Tanzu GemFire locator is a 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 cluster.
- *Server locators* give clients connection information to servers running in the locator's cluster. Server locators also monitor server load and send clients to the least-loaded servers.

By default, locators run as both peer and server locators.

Start a Locator with gfsh

This example uses the gfsh start locator command to start a default locator named "locator1":

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

When starting multiple locators, use parameters to give each a unique name, bind-address, and port to avoid conflicts with other locators. For example, this command starts "locator2" with a unique address and port:

```
gfsh> start locator --name=locator2 --bind-address=192.0.2.0 --port=13489
```

See the gfsh start locator reference page for command syntax.

Do not start locators in parallel (in other words, simultaneously). As a best practice, you should wait approximately 30 seconds for the first locator to complete startup before starting any other locators.

Locator Configuration and Log Files

Locator configuration and log files have the following properties:

- Locators are members of the cluster just like any other member. A locator should be configured in the same manner as a server. Therefore, if there are two other locators in the cluster, each locator should reference the other locators (just like a server member would). For example:

  ```
  gfsh> start locator --name=locator1 --port=9009 \
  --locators='host1[9001],host2[9003]'
  ```

- You can configure locators within the gemfire.properties file 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 cluster and the same gfsecurity.properties settings if you are using a separate, restricted access security settings file.

  For example, to configure locators in gemfire.properties:

  ```
  locators=host1[9001],host2[9003]
  ```

- There is no cache configuration specific to locators.
- 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 log file is automatically renamed, for example, locator1-01-01.log becomes locator1-02-01.log). You can modify the level of logging details in this file by specifying the desired level with the --log-level parameter.
- By default, the locator starts in a subdirectory, named after the locator, under the current working directory of the gfsh process. This subdirectory is considered the locator's current working directory. You can also specify a different working directory with the --dir parameter.

- By default, a locator that has been shut down and disconnected due to a network partition event or member unresponsiveness restarts and tries to reconnect to the existing cluster. When a locator is in the reconnecting state, it provides no discovery services for the cluster. See Handling Forced Cache Disconnection Using Auto-reconnect for more details.

Locators and the Cluster Configuration Service

Locators use the cluster configuration service to save configurations that apply to all cluster members, or to members of a specified group. The configurations are saved in the Locator's directory and are propagated to all locators in a cluster. See Overview of the Cluster Configuration Service. Servers receive the group-level and cluster-level configurations from the locators.

Check Locator Status

Once connected to the cluster with gfsh, check the status of a running locator by providing the locator name:

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

If you are not connected to a cluster, you can check the status of a local locator by providing the process ID, the locator's host name 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.

The command returns the following information, with the JVM arguments that were provided at startup:

```
$ gfsh status locator --dir=locator1
Locator in /Users/workspace/locator1 on 192.168.129.113[10334] as locator1 is currently online.
Process ID: 46075
Uptime: 55 seconds
VMware GemFire Version: 10.1.0
Java Version: 11.0.17
Log File: /Users/workspace/locator1/locator1.log
JVM Arguments:
...
```

Stop the Locator

When connected to the cluster in gfsh, stop a running locator by providing the locator name:

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

If not connected, 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.

Locators and Multi-Site (WAN) Deployments

If you use a multi-site (WAN) configuration, you can connect a locator to a remote site when starting the locator.

To connect a new locator process to a remote locator in a WAN configuration, specify the addresses of remote locators at startup:

```
gfsh> start locator --name=locator1 --port=9009 \
--J='-Dgemfire.remote-locators=192.0.2.0[9009],198.51.100.0[9009]'
```

Start a Locator with Custom Shell Script

You can start a locator, without using gfsh, using a custom shell script. If you have a script created for a previous version of Tanzu GemFire, update the script to use the new gemfire-bootstrap.jar. When you create a script, do not include any other jars on the java -classpath argument other than the gemfire-bootstrap.jar. VMware recommends that you avoid wildcards or other globbing patterns that may unintentionally include other jars. Use the --automatic-module-path bootstrap argument to include any custom classes at startup.

For an example script, see the start-locator script in the gemfire-examples repository in GitHub.

Start a Locator Programmatically

Running your locators standalone, as described above, provides the highest reliability and availability of the locator service as a whole. But in some situations, you may choose to run an embedded locator.

To start the locator inside your code, use the org.apache.geode.distributed.LocatorLauncher API. Use the LocatorLauncher.Builder class to construct an instance of the LocatorLauncher, and then use the start() method to start the locator service. The other methods in the LocatorLauncher class provide status information about the locator and allow you to stop the locator.

```
package com.example;

import org.apache.geode.distributed.LocatorLauncher;

public class CustomLocatorApplication {
  public static void main(final String[] args) {
    final LocatorLauncher locatorLauncher = new LocatorLauncher.Builder()
        .setMemberName(args[0])
        .setPort(Integer.parseInt(args[1]))
        .build();

    locatorLauncher.start();

    System.out.println("Locator successfully started");
  }
}
```

This example demonstrates how to start a GemFire locator embedded in your own Java application via the LocatorLauncher utilizing the GemFire Bootstrap. Using the GemFire Bootstrap is necessary to load GemFire Extensions, like GemFire Search, and isolate GemFire from other loaded application classes. When using the GemFire Bootstrap, you must be sure to define a GEMFIRE_HOME environment variable or a gemfire.home system property that points to the Tanzu GemFire top-level installation directory.

The class com.example.CustomLocatorApplication has a static void main(String[]) method, like any other Java application. It uses the org.apache.geode.distributed.LocatorLauncher to embed a GemFire locator in the Java application.

The Java application is executed through GemFire Bootstrap's com.vmware.gemfire.bootstrap.Main Java application class. The command includes only the GemFire Bootstrap jar in the Java classpath. Your application classes and jars can be added via the --automatic-module-classpath argument. You should not include any other jars in the Java classpath directly.

```
$ java -classpath gemfire-bootstrap.jar com.vmware.gemfire.bootstrap.Main \
    com.example.CustomLocatorApplication \
    --automatic-module-classpath <classes:jar:...> \
    name 1234 [other application arguments ...]
```

The full example can be found as CustomLocatorApplication.java in the gemfire-examples repository in GitHub. start-locator.sh is an example shell script for running Java with the a custom locator launcher.

Running GemFire Server Processes

This topic explains how to run VMware Tanzu GemFire server processes.

A Tanzu GemFire server is a process that runs as a long-lived, configurable member of a client/server 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 servers using the gfsh command-line tool.

Start a Server with gfsh

Assuming that you have already used gfsh to start a locator, this example uses the gfsh start server command to start a default server named "server1":

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

See the gfsh start server reference page for command syntax.

Default Server Configuration and Log Files

The gfsh utility uses a working directory for its configuration files and log files. These are the defaults and configuration options:

- A server is configured like any other GemFire process, with gemfire.properties and shared cluster configuration files. It is not programmable except through application plug-ins. Typically, you provide the gemfire.properties file and the gfsecurity.properties file. You can also specify a cache.xml file in the cache server's working directory.
- A new server receives its initial cache configuration from the cluster configuration service, assuming the locator is running the cluster configuration service (the default). The shared configuration consists of cache.xml files, gemfire.properties files, and deployed jar files. You can

deactivate use of the cluster configuration service by specifying the --use-cluster-configuration=false parameter. See Overview of the Cluster Configuration Service.

- If you are using the Spring Framework, you can specify a Spring ApplicationContext XML file when starting up your server by using the --spring-xml-location parameter. This parameter allows you to bootstrap your server process with your Spring application's configuration. See Spring documentation for more information about this file.
- Log file output defaults to server-name.log in the cache server's working directory. If you restart a server with a previously used server name, the existing log file is automatically renamed, for example, server1-01-01.log becomes server1-02-01.log. You can modify the level of logging details in this file by specifying the desired level with the --log-level parameter.
- By default, the server starts in a subdirectory, named after the server, under the current working directory of the gfsh process. This subdirectory is considered the server's current working directory. You can also specify a different working directory with the --dir parameter.
- By default, a server process that has been shut down and disconnected due to a network partition event or member unresponsiveness restarts and tries to reconnect to the existing cluster. See Handling Forced Cache Disconnection Using Auto-reconnect for more details.
- You can pass JVM parameters to the server's JVM with the --J parameter. The JVM parameters you specify can be any Java command-line argument including, but not limited to, system properties such as gemfire.jmx-manager. For example:

  ```
  gfsh>start server --name=server1 --J=-Dgemfire.jmx-manager=true \
  --J=-Dgemfire.jmx-manager-start=true --J=-Dgemfire.http-port=8080
  ```

- When both --max-heap and --initial-heap are specified during server startup, additional GC parameters are specified on your behalf. If you do not want additional default GC properties set, then use the -Xms and -Xmx JVM options to set only these parameters. For more information, see Controlling Heap Use with the Resource Manager in *Managing Heap Memory*.

  To start a server providing GC configuration settings:

  ```
  gfsh>start server --name=server3 \
  --J=-Xms80m,-Xmx80m,-XX:+UseConcMarkSweepGC,-XX:CMSInitiatingOccupancyFraction=65
  ```

- We recommend that you do not use the -XX:+UseCompressedStrings and -XX:+UseStringCache JVM configuration properties when starting up servers. These JVM options can cause issues with data corruption and compatibility.

Check Server Status

Once connected to the cluster in gfsh, check the status of a running cache server by providing the server name:

```
gfsh>status server --name=server1
```

If you are not connected to a cluster, 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=server1
```

If successful, the output provides information as in this sample:

```
% gfsh status server --dir=server4
Server in /home/username/server4 on 192.0.2.0[40404] as server4 is currently online.
Process ID: 49008
Uptime: 2 minutes 4 seconds
Tanzu GemFire Version: 10.2
Java Version: 1..0_361
Log File: /home/username/server4/server4.log
JVM Arguments:
...
```

Stop a Server

When connected to the cluster in gfsh, stop a running cache server by providing the server name:

```
gfsh>stop server --name=server1
```

If not connected, 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=server1
```

You can also use the gfsh shutdown command to shut down all cache servers in an orderly fashion. Doing a shutdown is the correct approach for systems with persistent regions. See Starting Up and Shutting Down Your System for more details.

Start a Server with Custom Shell Script

You can start a server, without using gfsh, using a custom shell script. If you have a script created for a previous version of Tanzu GemFire, update the script to use the new gemfire-bootstrap.jar. When you create a script, do not include any other jars on the java -classpath argument other than the gemfire-bootstrap.jar. VMware recommends that you avoid wildcards or other globbing patterns that may unintentionally include other jars. Use the --automatic-module-path bootstrap argument to include any custom classes at startup.

For an example script, see the start-server script in the gemfire-examples repository in GitHub.

Start a Server Programmatically

Running your servers standalone, as described above, provides the highest reliability and availability. But in some situations, you may choose to run an embedded server.

To start the server programmatically use the org.apache.geode.distributed.ServerLauncher API to start the cache server process inside your code. Use the ServerLauncher.Builder class to construct an instance of the ServerLauncher, and then use the start() method to start the server service. The other methods in the ServerLauncher class provide status information about the server and allow you to stop the server.

```
package com.example;

import org.apache.geode.distributed.ServerLauncher;

public class CustomServerApplication {
  public static void main(final String[] args) {
    final ServerLauncher serverLauncher = new ServerLauncher.Builder()
        .setMemberName(args[0])
        .setServerPort(Integer.parseInt(args[1]))
        .build();

    serverLauncher.start();

    System.out.println("Server successfully started");
  }
}
```

This example demonstrates how to start a GemFire server embedded in your own Java application via the ServerLauncher utilizing the GemFire Bootstrap. Using the GemFire Bootstrap is necessary to load GemFire Extensions, like GemFire Search, and isolate GemFire from other loaded application classes. When using the GemFire Bootstrap, you must be sure to define a GEMFIRE_HOME environment variable or a gemfire.home system property that points to the Tanzu GemFire top-level installation directory.

The class com.example.CustomServerApplication has a static void main(String[]) method, like any other Java application. It uses the org.apache.geode.distributed.ServerLauncher to embed a GemFire server in the Java application.

The Java application is executed through GemFire Bootstrap's com.vmware.gemfire.bootstrap.Main Java application class. The command includes only the GemFire Bootstrap jar in the Java classpath. Your application classes and jars can be added via the --automatic-module-classpath argument. You should not include any other jars in the Java classpath directly.

```
$ java -classpath gemfire-bootstrap.jar com.vmware.gemfire.bootstrap.Main \
    com.example.CustomServerApplication \
    --automatic-module-classpath <classes:jar:...> \
    name 1235 [other application arguments ...]
```

The full example can be found as CustomServerApplication.java in the gemfire-examples repository in GitHub. start-server.sh is an example shell script for running Java with the a custom server launcher.

Managing System Output Files

This topic explains how to manage VMware Tanzu GemFire system output files.

Tanzu GemFire output files are optional and can become excessively large. Work with your system administrator to determine where to place them to avoid interfering with other system activities.

Tanzu 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.
- 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 the Reference.
- 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.

Firewall Considerations

This topic provides link to documentation about firewall considerations when using VMware Tanzu GemFire.

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.

- Firewalls and Ports

  Make sure your port settings are configured correctly for firewalls.

Firewalls and Connections

This topic discusses possible connection issues that can result from running a firewall on a system when using VMware Tanzu GemFire.

Tanzu 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 Tanzu 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.

For more information about how Tanzu GemFire client and servers connect, see the following topics:

- How Client/Server Connections Work
- Socket Communication
- Controlling Socket Use
- Setting Socket Buffer Sizes

Firewalls and Ports

This topic explains how to ensure that your port settings are configured correctly for firewalls when using VMware Tanzu GemFire.

There are several different port settings that need to be considered when using firewalls:

- Port that the cache server listens on. This is configurable using the cache-server element in cache.xml, on the CacheServer class in Java APIs, and as a command line option to the gfsh start server command.

  By default, if not otherwise specified, Tanzu GemFire clients and servers discover each other on a pre-defined port (40404) on the localhost.

- Locator port. Tanzu GemFire clients can use the locator to automatically discover cache servers. The locator port is configurable as a command-line option to the gfsh start locator command. Locators are used in the peer-to-peer cache deployments 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.

  By default, if not otherwise specified, Tanzu GemFire locators use the default port 10334.
- Since locators start up the cluster, locators must also have their ephemeral port range and TCP port accessible to other members through the firewall.
- For clients, you configure the client to connect to servers using the client's pool configuration. The client's pool configuration has two options: 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. The ports specified must be made accessible through your firewall.

Limiting Ephemeral Ports for Peer-to-Peer Membership

By default, Tanzu 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 the TCP port for each process.

Properties for Firewall and Port Configuration

This table contains properties potentially involved in firewall behavior, with a brief description of each property.

Configuration area	Property or Setting	Definition
peer-to-peer config	conserve-sockets	Specifies whether sockets are shared by the system member's threads.
peer-to-peer config	locators	The list of locators used by system members. The list must be configured consistently for every member of the cluster.
peer-to-peer config	membership-port-range	The range of ephemeral ports available membership communication and TCP failure detection in the peer-to-peer cluster.
peer-to-peer config	tcp-port	The TCP port to listen on for cache communications.
Configuration Area	**Property or Setting**	**Definition**
cache server config	hostname-for-clients	Hostname or IP address to pass to the client as the location where the server is listening.

Configuration Area	Property or 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 (cache.xml) or --port parameter to the gfsh start server command	Port that the server listens on for client communication.

Default Port Configurations

Port Name	Related Configuration Setting	Default Port
Cache Server	port (cache.xml)	40404
HTTP	http-service-port	7070
Locator	start-locator (for embedded locators) or --port parameter to the gfsh start locator command.	*if not specified upon startup or in the start-locator property, uses default port 10334*
Membership Port Range	membership-port-range	41000 to 61000
Memcached Port	memcached-port	*not set*
RMI	jmx-manager-port	1099
TCP	tcp-port	ephemeral port

Properties for Firewall and Port Configuration in Multi-Site (WAN) Configurations

Each gateway receiver uses a single port to accept connections from gateway senders in other systems. The configuration of a gateway receiver specifies a range of possible port values to use. Tanzu 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.

Configuration Area	Property or Setting	Definition
multi-site (WAN) config for gateway sender	hostname-for-senders	Hostname or IP address of the gateway receiver used by gateway senders to connect.
multi-site (WAN) config for locator	remote-locators	List of locators (and their ports) that are available on the remote WAN site.
multi-site (WAN) config for gateway receiver	start-port and end-port (cache.xml) or –start-port and –end-port parameters to the gfsh start gateway receiver command	Port range that the gateway receiver can use to listen for gateway sender communication.

Basic Configuration and Programming

Basic Configuration and Programming describes how to configure cluster and cache properties for your VMware Tanzu GemFire installation. For your applications, it provides guidance for writing code to manage your cache and cluster connection, data regions, and data entries, including custom classes.

- Cluster and Cache Configuration

  To work with your VMware Tanzu GemFire applications, you use a combination of configuration files and application code.

- Cache Management

  The Tanzu GemFire cache is the entry point to Tanzu GemFire caching management. Tanzu GemFire provides different APIs and XML configuration models to support the behaviors of different members.

- Data Regions

  The region is the core building block of the VMware Tanzu GemFire cluster. All cached data is organized into data regions and you do all of your data puts, gets, and querying activities against them.

- 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 VMware Tanzu GemFire requirements for data storage and distribution.

Distributed System and Cache Configuration

This topic provides links to documentation about VMware Tanzu GemFire cluster and cache configuration. To work with your Tanzu GemFire applications, you use a combination of configuration files and application code.

- Cluster Members

  Cluster members are programs that connect to a Tanzu GemFire cluster. You configure members to belong to a single cluster, and you can optionally configure them to be clients or servers to members in other clusters, and to communicate with other clusters.

- Setting Properties

  Tanzu GemFire provides a default cluster configuration for out-of-the-box systems. To use non-default configurations and to fine-tune your member communication, you can use a mix of various options to customize your cluster configuration.

- Options for Configuring the Cache and Data Regions

  To populate your VMware Tanzu 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.

- Local and Remote Membership and Caching

For many VMware Tanzu GemFire discussions, you need to understand the difference between local and remote membership and caching.

Cluster Members

This topic defines and explains VMware Tanzu GemFire cluster members.

Cluster members are programs that connect to a Tanzu GemFire cluster. You configure members to belong to a single cluster, and you can optionally configure them to be clients or servers to members in other clusters, and to communicate with other clusters.

Member Overview

Cluster members (or simply “members”) connect to the Tanzu GemFire cluster when they create the Tanzu GemFire data cache. The members’ cluster is configured through Tanzu GemFire properties. See gemfire.properties and gfsecurity.properties (Tanzu GemFire Properties). Tanzu GemFire properties specify all necessary information for member startup, initialization, and communication.

> You cannot change a member’s properties while the member is connected to the cluster.

Use the properties to define:

- How to find and communicate with other members
- How to perform logging and statistics activities
- Which persistent configuration or cache.xml file to use for cache and data region initialization
- Other options, including event conflation, how to handle network loss, and security settings

Membership and System Topologies

Every Tanzu GemFire process is a member of a cluster, even if the cluster is defined as standalone, with just one member. You can run an individual cluster in isolation or you can combine clusters for vertical and horizontal scaling. See Topology and Communication General Concepts.

- Peer-to-Peer Clusters. Members that define the same member discovery properties belong to the same cluster and are peers to one another.
- Client/Server Installations. The client/server topology uses relationships that you configure between members of multiple clusters. You configure some or all of the peers in one cluster to act as cache servers to clients connecting from outside the cluster. 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 Tanzu GemFire clusters, 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 clusters. Through this configuration, you loosely couple two or more clusters for automated data distribution. This is usually done for sites at geographically separate locations. You configure a subset of peers in each cluster site with gateway senders and/or gateway receivers to manage events that are distributed between the sites.

In the context of a single cluster, unless otherwise specified, “remote member” refers to another member of the same cluster. In client/server and multi-site installations, “remote” generally refers to members in other clusters. 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 clusters, and to those gateway senders’ peers.

Setting Properties

This topic explains how to customize your VMware Tanzu GemFire configuration by setting properties.

Tanzu GemFire provides a default configuration for out-of-the-box systems. To use non-default configurations and to fine-tune your member communication, you can use a mix of various options to customize your configuration.

Tanzu GemFire properties are used to join a cluster and configure system member behavior. Configure your Tanzu 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 a username and password that you have received from keyboard input.

Check with your Tanzu GemFire system administrator before changing properties through the API, including the gemfire.properties and gfsecurity.properties 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.

The product defaultConfigs directory has a sample gemfire.properties file with all default settings.

Set 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("gemfirePropertyFile", "gfTest");

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

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

     ```
     java -DgemfirePropertyFile=gfTest 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 files on the gfsh command line as command-line options. Example:

```
gfsh>start server --name=server_name --properties-file=serverConfig/gemfire.properties --security-propertie
s-file=gfsecurity.properties
```

  See Running Tanzu GemFire Server Processes for more information about running cache servers.

3. Entry in a gemfire.properties file. See Deploying Configuration Files without the Cluster Configuration Service. Example:

```
cache-xml-file=cache.xml
conserve-sockets=true
```

4. Default value. The default values are defined within the API for org.apache.geode.distributed.ConfigurationProperties.

Options for Configuring the Cache and Data Regions

This topic discusses options for configuring VMware Tanzu GemFire cache and data regions.

To populate your Tanzu GemFire cache and fine-tune its storage and distribution behavior, you must 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, and settings designating the member as a server
- Cache data regions

Configure the cache and its data regions through one or more of these methods:

- Through a persistent configuration that you define when issuing commands that use the gfsh command line utility. The gfsh utility supports the administration, debugging, and deployment of VMware Tanzu GemFire processes and applications. You can use gfsh to configure regions, locators, servers, disk stores, event queues, and other objects.

  As you issue commands, gfsh saves a set of configurations that apply to the entire cluster and also saves configurations that only apply to defined groups of members within the cluster. You can re-use these configurations to create a cluster. See Overview of the Cluster Configuration Service.

- 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.
- Through application calls to the org.apache.geode.cache.CacheFactory, org.apache.geode.cache.Cache and org.apache.geode.cache.Region APIs.

Local and Remote Membership and Caching

This topic describe the difference between local and remote membership and caching in VMware Tanzu GemFire.

Discussions of Tanzu 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 cluster or in different clusters.

Cache Management

This topic provides links to documentation about VMware Tanzu GemFire cache management.

The Tanzu GemFire cache is the entry point to Tanzu GemFire caching management. Tanzu 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.

- 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.

- 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.

- 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.

- 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 Tanzu 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.

- Launching an Application after Initializing the Cache

  You can specify a callback application that is launched after the cache initialization.

Introduction to Cache Management

This topic introduces cache management in VMware Tanzu GemFire.

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 org.apache.geode.cache.Cache.

You generally configure caches using the gfsh command-line utility or a combination of XML declarations and API calls. Tanzu GemFire loads and processes your XML declarations when you first create the cache.

Tanzu GemFire has one cache type for managing server and peer caches and one for managing client caches. The cache server 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

Tanzu GemFire's caching APIs provide specialized behavior for different system member types and security settings.

- org.apache.geode.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.
- org.apache.geode.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 cluster. GemFireCache is implemented by Cache and ClientCache.
- org.apache.geode.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 cluster, region creation, transactions and querying, and cache server functionality.
- org.apache.geode≈setting_cache_initializer.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 product XML schema definition cache-1.0.xsd. The schema definition file is available at http://geode.apache.org/schema/cache/cache-1.0.xsd.

You use one format for peer and server caches and another for client caches.

cache.xml for Peer/Server:

```
<?xml version="1.0" encoding="UTF-8"?>
<cache xmlns="http://geode.apache.org/schema/cache"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cache-1.0.xsd"
   version="1.0">
...
</cache>
```

cache.xml for Client:

```
<?xml version="1.0" encoding="UTF-8"?>
<client-cache
   xmlns="http://geode.apache.org/schema/cache"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cache-1.0.xsd"
   version="1.0">
...
</client-cache>
```

For more information about the cache.xml file, see cache.xml.

Create and Close a Cache

Your system configuration and cache configuration are initialized when you start your member processes and create each member's Tanzu GemFire cache. If you are using the cluster configuration service, member processes can pick up its cache configuration from the cluster or group's current configuration. See Overview of the Cluster Configuration Service.

The steps in this section use gemfire.properties and cache.xml file examples, except where API is required. You can configure your cluster properties and cache through the API as well, and you can use a combination of file configuration and API configuration.

The XML examples may not include the full cache.xml file listing. All of your declarative cache configuration must conform to the cache XSD at http://geode.apache.org/schema/cache/cache-1.0.xsd.

For all of your Tanzu GemFire applications:

1. Create your Cache, for peer/server applications, or ClientCache, for client applications. This connects to the Tanzu 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 cluster in an orderly manner.

Follow the instructions in the subtopics under Cache Management 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.

Cache Management with gfsh and the Cluster Configuration Service

You can use gfsh commands to mange a server cache. There are gfsh commands to create regions, start servers, and to create queues and other objects. As you issue these commands, the Cluster Configuration Service saves cache.xml and gemfire.properties files on the locators and distributes those configurations to any new members that join the cluster. See Overview of the Cluster Configuration Service.

Managing a Peer or Server Cache

This topic explains how to manage a peer or server cache in VMware Tanzu GemFire.

You start your peer or server cache using a combination of XML declarations and API calls. Close the cache when you are done.

Tanzu GemFire peers are members of a Tanzu GemFire cluster that do not act as clients to another Tanzu GemFire cluster. Tanzu GemFire servers are peers that also listen for and process client requests.

1. Create your cache:
    1. Start up a cluster and the cluster configuration service:
        1. Start a locator with --enable-cluster-configuration set to true. (It is set true by default.)

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

        2. Start up member processes that use the cluster configuration service (enabled by default):

            ```
            gfsh>start server --name=server1 --server-port=40404
            ```

        3. Create regions:

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

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

    2. Or if you are not using the cluster configuration service, directly configure cache.xml in each member of your cluster. In your cache.xml, use the cache DOCTYPE and configure your cache inside a <cache> element. Example:

        ```
        <?xml version="1.0" encoding="UTF-8"?>
        <cache
           xmlns="http://geode.apache.org/schema/cache"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cac
        he-1.0.xsd"
           version="1.0">
           // 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>
        ```

    3. To programmatically 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 Tanzu GemFire cacheserver process, it automatically creates the cache and connection at startup and closes both when it exits.

        The system creates the connection and initializes the cache according to your gemfire.properties and cache.xml specifications.

2. Close your cache when you are done using the inherited close method of the Cache instance:

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

Managing a Client Cache

This topic explains how to manage a client cache in VMware Tanzu GemFire.

Several options exist 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.

Tanzu GemFire clients are processes that send most or all of their data requests and updates to a Tanzu GemFire server system. Clients run as standalone processes, without peers of their own.

Tanzu GemFire automatically configures the cluster for your ClientCache as standalone, which means the client has no peers. Do not try to set the gemfire.properties locators for a client application or the system will throw an exception.

1. Create your client cache:

    1. 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:

    ```
    <?xml version="1.0" encoding="UTF-8"?>
    <client-cache
       xmlns="http://geode.apache.org/schema/cache"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cac
    he-1.0.xsd"
       version="1.0">
       <pool name="serverPool">
          <locator host="host1" port="44444"/>
       </pool>
       <region name="exampleRegion" refid="PROXY"/>
    </client-cache>
    ```

    > 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. Tanzu 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 unsynchronized with the server cache. See Consistency for Region Updates.

    2. 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"/>
```

3. 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

This topic discusses how to manage a cache in a secure system in VMware Tanzu GemFire.

A secured system does both authentication at connection time and authorization prior to cache operations. Client apps and cluster members (servers and locators) require configuration and setup when the SecurityManager is enabled.

For more details, see Security. For information about authentication, see Implementing Authentication.

Managing RegionServices for Multiple Secure Users

This topic explains how to manage RegionService for multiple secure users in VMware Tanzu GemFire.

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 Tanzu 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.

1. Create your cache and RegionService instances:
    1. 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 deactivates it for the ClientCache instance.

2. 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.

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.

Launching an Application after Initializing the Cache

This topic explains how to launch an application after initializing the cache in VMware Tanzu GemFire.

The initializer feature in GemFire allows you to specify an application that runs automatically after cache initialization. This allows for configuring resources, starting background processes, or setting up custom monitoring tools, ensuring your application environment is ready as soon as the cache is initialized.

Implementing an Initializer Class

To use an initializer, create a class that defines the actions to run after cache initialization.

The class must implement either the Declarable interface or the CacheCallback interface:

- Declarable: Use this interface if no actions are needed during shutdown. Its initialize method is called when the class is loaded, with any parameters passed as properties.

- CacheCallback: Use this interface to execute shutdown actions. It provides the close method, called when the initializer is destroyed via the destroy initializer command. Note that this does not occur during server shutdown since all processes terminate.

Example Class Definition

```
import org.apache.geode.cache.Declarable;
import java.util.Properties;

public class MyInitializer implements Declarable {
  public void init(Properties properties) {
    System.out.println(properties.getProperty("members"));
  }
}
```

Configuring an Initializer

You can configure an initializer in two ways:

1. Using the <initializer> tag in your cache.xml file.
2. Using the create initializer command in gfsh.

Configuring an Initializer in cache.xml

To set up an initializer in your cache.xml file:

- Use the <initializer> element to specify the callback class.
- Place this element at the end of your cache.xml file.
- The <initializer> element can be used for both server and client caches.

Example cache.xml Configuration:

```
<initializer>
  <class-name>MyInitializer</class-name>
    <parameter name="members">
      <string>2</string>
    </parameter>
</initializer>
```

Applications that start a server using the cache server script can use this feature to hook into a callback application.

Configure and Manage Initializers using gfsh

In GemFire, you can configure and manage initializers using the create initializer, list initializers, and destroy initializer commands. Note that servers must have cluster configuration enabled to use these commands.

Create Initializer Use the create initializer command to define a new initializer and add it to the cluster configuration.

``pre create initializer –initializer=MyInitializer{“members”:“2”}

```
When creating an initializer:
   - Provide the initializer class name using the `--initializer` option. Append any parameters in JSON format, if required.
   - Initializers can be configured on a per-group basis. If no group is specified, the initializer applies to all servers in the `cluster` group. There can only be one initializer per group.

How the initializer is invoked depends on the state of the servers:
1. **If servers are not running:** The initializer will be applied during the next server startup.
2. **If servers are running and `--run-on-create` is included:** The initializer will run immediately and will also execute during future server restarts.
3. **If servers are running and `--run-on-create` is not included:** The initializer will be configured but will only execute during the next server restart.

**List Initializers**
This command shows all configured initializers as well as the members with which the initializer is configured.

**Destroy Initializers**
Use this command to remove an initializer that is no longer required. If the `CacheCallback` method has been implemented, it will be invoked at this time.

## Real-world Usage Scenarios
Below are two common ways initializers can be utilized in real-world applications:

1.  Start a SystemMembershipListener

   ```pre
   <initializer>
     <class-name>TestSystemMembershipListener</class-name>
   </initializer>
   ```

2. Write a custom tool that monitors cache resources

   ```pre
   <initializer>
     <class-name>ResourceMonitorCacheXmlLoader</class-name>
   </initializer>
   ```

Any singleton or timer task or thread can be instantiated and started using the initializer element.
```

Data Regions

This topic provides links to documentation about VMware Tanzu GemFire data regions.

The region is the core building block of the Tanzu GemFire cluster. All cached data is organized into data regions and all of your data puts, gets, and querying activities are performed against them.

- Region Management

  VMware Tanzu GemFire provides gfsh commands, APIs, and XML configuration models to support the configuration and management of data regions.

- Region Naming

  To be able to perform all available operations on your data regions, follow these region naming guidelines.

- Region Shortcuts and Custom Named Region Attributes

Tanzu 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.

- Storing and Retrieving Region Shortcuts and Custom Named Region Attributes

  Use these examples to get started with Tanzu GemFire region shortcuts.

- Managing Region Attributes

  Use region attributes to fine-tune the region configuration provided by the region shortcut settings.

- 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.

- Building a New Region with Existing Content

  A new region or cluster may need to be loaded with the data of an existing system. There are two approaches to accomplish this task. The approach used depends upon the organization of both the new and the existing cluster.

Region Management

This topic explains how to manage regions in VMware Tanzu GemFire.

Operations that create, destroy, invalidate, clear, and change the configuration of regions work with gfsh commands, through an XML description, and using API calls.

You store your data in region entry key/value pairs, with keys and values being any object types your application needs. The org.apache.geode.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.

Region shortcuts identify commonly-used types of regions. See Region Shortcuts for more information.

> If you change attributes that define a region, you must restart the member for the changes to take effect.

Creating a Region

Creating a Region with gfsh

A common way to create a data region in the VMware Tanzu GemFire cache is to use the gfsh command-line tool.

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 gfsh create region command reference page details command line options for creating a region with gfsh.

With gfsh connected to a JMX server, an example command that creates a replicated region is

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

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 for details.

> The cluster configuration service, which is enabled by default, automatically saves the configuration on the locators in the cluster. After you use the gfsh create region command, any new servers that you start that attach to the same locator receive the same configuration. You can also create alternate configurations within a cluster by specifying a group when creating the region and starting servers. See Overview of the Cluster Configuration Service.

Creating a Region Through the cache.xml File

A common way to create a data region in the VMware Tanzu GemFire cache is through cache.xml declarations. When starting the member with the cache.xml file, the region will be created.

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.

- In the cache.xml file, create a <region> element for the new region as a subelement to the <cache> element or the <client-cache> element.
- Define the region's name and use a region shortcut, if one applies.
- Add other attributes as needed to customize the region's behavior.

cache.xml File Examples

The region declaration of a replicated region named Portfolios:

```
<region name="Portfolios" refid="REPLICATE"/>
```

The region declaration of a partitioned region named myRegion:

```
<region name="myRegion" refid="PARTITION"/>
```

The region declaration of a partitioned region that backs up content to disk:

```
<region name="myRegion" refid="PARTITION_PERSISTENT"/>
```

The region declaration of a partitioned region configured with high availability and a modified storage capacity in the host member:

```
<region name="myRegion" refid="PARTITION_REDUNDANT">
  <region-attributes>
    <partition-attributes local-max-memory="512" />
  </region-attributes>
</region>
```

The region declaration of a replicated region configured with an event listener in which entries expire:

```
<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

Tanzu 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 org.apache.geode.cache package. They correspond to cache.xml declarations within the <cache> element for creating and configuring regions.
    - org.apache.geode.cache.Cache.createRegionFactory . This method takes a RegionShortcut enum to initiate region configuration, and it returns a RegionFactory. Use createRegionFactory(), not new RegionFactory, to create a RegionFactory.
    - org.apache.geode.cache.RegionFactory. Provides methods to set individual region attributes and to create the region. The create call returns a Region.
    - org.apache.geode.cache.RegionShortcut. Defines common region configurations.
- Client Region APIs. Use these methods, interfaces, and classes for client region creation. These are in the org.apache.geode.cache.client package. They correspond to cache.xml 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.
    - org.apache.geode.cache.clientCache.createRegionFactory . This method takes a ClientRegionShortcut enum to initiate region configuration, and returns a ClientRegionFactory.
    - org.apache.geode.cache.client.ClientRegionFactory. Provides methods to set individual region attributes and to create the region. The create call returns Region.
    - org.apache.geode.cache.client.ClientRegionShortcut . Defines common region configurations.
- Region APIs Used For All Member Types. These interfaces and classes are used universally for region management. These are in the org.apache.geode.cache package. They correspond to cache.xml declarations in the <cache> and <client-cache> elements for creating and configuring regions.
    - org.apache.geode.cache.Region . Interface for managing regions and their entries.
    - org.apache.geode.cache.RegionAttributes . Object holding region configuration settings.

Use the API to create regions in the cache after startup. For run-time region creation, you need to use the API.

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.

1. Use a region shortcut to create your region factory.
    - In peers and servers, use org.apache.geode.cache.RegionFactory.
    - In clients, use org.apache.geode.cache.client.ClientRegionFactory.

2. (Optional) Use the region factory to further configure your region.
3. Create your region from the configured region factory.

API Examples

Create a replicated region named Portfolios:

```
Cache cache = CacheFactory.create();
RegionFactory rf = cache.createRegionFactory(REPLICATE);
Region pfloRegion = rf.create("Portfolios");
```

Create a partitioned region with a listener:

```
RegionFactory rf =
   cache.createRegionFactory(RegionShortcut.PARTITION);
rf.addCacheListener(new LoggingCacheListener());
custRegion = rf.create("customer");
```

Create a partitioned region with a partition resolver for colocated regions:

```
PartitionAttributesFactory<CustomerId, String> paf = new PartitionAttributesFactory<>();
paf.setPartitionResolver(new CustomerOrderResolver());

RegionFactory rf =
   cache.createRegionFactory(RegionShortcut.PARTITION);
rf.setPartitionAttributes(paf.create());
rf.addCacheListener(new LoggingCacheListener());
custRegion = rf.create("customer");
```

Create a client region with a pool specification:

```
ClientRegionFactory<String,String> cRegionFactory =
   cache.createClientRegionFactory(PROXY);
Region<String, String> region =
   cRegionFactory.setPoolName("Pool3").create("DATA");
```

Create and Access Data Subregions

An individual region can contain multiple subregions. Subregions are an older feature that will not be useful in new designs and applications. They are used to create a hierarchical namespace within a cache, providing naming that feels like paths in a file system. Here are limitations on the use of subregions:

- A region with LOCAL scope can only have subregions with LOCAL scope.
- Partitioned region types may not be used with subregions. A subregion may not have a parent that is a partitioned region, and a subregion may not be of type PARTITION.
- A subregion must have the same scope (GLOBAL, DISTRIBUTED_ACK, DISTRIBUTED_NO_ACK) as its parent region.
- Subregion names must be unique within the cache.

You can create subregions using one of the following methods:

- Declaration in the cache.xml:

```
<?xml version="1.0"?>
<cache
   xmlns="http://geode.apache.org/schema/cache"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cache-1.0.xs
d"
   version="1.0"
   lock-lease="120"
   lock-timeout="60"
   search-timeout="300">
<!-- Create a region named Portfolios -->
  <region name="Portfolios" refid="REPLICATE">
     <region name="Private" refid="REPLICATE">
     ...
     </region>
  </region>
</cache>
```

When the cache.xml is loaded at cache creation, the system automatically creates any declared regions and subregions.

- RegionFactory API calls:

```
Cache cache = CacheFactory.create();
RegionFactory rf = cache.createRegionFactory(REPLICATE);
Region pfloRegion = rf.create("Portfolios");
Region pvtSubregion = rf.createSubregion(pfloRegion, "Private");
```

Region method calls with a recursive parameter operate on the given regions and then recursively on all contained subregions.

Update the Configuration of Data Regions

Update your region properties and contents through alter region command, the API or from cache.xml file declarations.

- Use the gfsh alter region command.
- 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 a Region

An invalidate region operation removes all entry values for a region, while leaving the entry keys intact. This operation can be invoked only through the API on a Region instance. Event notification occurs.

```
// Invalidate the entire distributed region
Region.invalidateRegion();
```

The API also offers a method to invalidate only the entries within the local cache. This method may not be used on a replicated region, as doing so would invalidate the replication contract.

```
// Invalidate the region within this member
Region.localInvalidateRegion();
```

Clear a Region

A clear region operation removes all entries from a region. This operation is not available for partitioned regions. This operation can be invoked through the API on a Region instance:

```
// Remove all entries for the region
Region.clear();
```

It can be invoked with the gfsh command:

```
gfsh>remove --region=Region1 --all
```

Event notification occurs for a clear region operation.

Destroy a Region

A destroy region operation removes the entire region. This operation can be invoked through the API on a Region instance:

```
// Remove the entire region
Region.destroyRegion();
```

A destroy region operation can be invoked with the gfsh command:

```
gfsh>destroy region --name=Region1
```

Event notification occurs for a destroy region operation.

A region can be destroyed by removing the region's specification from the cache.xml file.

Destroying the region by an API invocation or by using the gfsh destroy command while all members are online is the best way to remove a region, as Tanzu GemFire handles all aspects of the removal, including removing the region's persistent disk stores across the online members hosting the region. Destroying the region by removing its specification from the cache.xml file does not remove the region's existing persistent disk stores.

The destroy operation can be propagated only to online members. The system will encounter restart issues if a region is destroyed while some members are online and others are offline. As those members that were offline restart, they will block indefinitely, waiting for persistent region data that no longer exists. To fix this issue, shut down all members that are blocked waiting for the removed region. Once those members are in the offline state, use the gfsh alter disk-store command with the --remove option on each offline member to remove the region. Then, restart each member.

An edge case results in issues when destroying a persistent region (R-removed) by removing its specification from the cache.xml file, and region R-removed was colocated with another persistent region (R-remains). The issue occurs because the persistent information contained within R-remains is inconsistent with the (lack of) specification of R-removed. Upon restart of R-remains, its persisted metadata refers to R-removed as a colocated region, and the startup of R-remains is dependent on that removed region. Thus, the startup of R-remains blocks, unable to complete. The issue may manifest with operations on the R-remains region such as a query, put, or get, that never finishes. To fix this issue, shut down all members with the persisted metadata that refers to the removed region. Once those members

are in the offline state, use the gfsh alter disk-store command with the --remove option on each offline member to remove the region. Then, restart each member.

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.
- 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.

Region Naming

This topic provides guidlines for region naming in VMware Tanzu GemFire.

To be able to perform all available operations on your data regions, follow these region naming restrictions:

- The only characters permitted in region names by default are:
  - alphanumeric characters (ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789)
  - hyphens (-)
  - periods (.)
  - underscores (_).
- Region names are case sensitive.
- The slash character (/) is not allowed in region names.
- Your region names cannot begin with two underscore characters (__). This usage is reserved for internal use.
- The following characters are not allowed in region names: left brackets ([), right brackets (]), carats (^), backticks (</code>), backslashes (). To use any of these special characters in region names, set the value of the GemFire system propertygemfire.allowSpecialRegionNameCharacterstotrue`.

Region Shortcuts and Custom Named Region Attributes

This topic explains how to use region shortcuts and custom named region attributes in VMware Tanzu GemFire.

Tanzu 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.

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, reliability behavior, and the automatic loading, distribution, and expiration of data entries.

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.

Tanzu 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 Tanzu GemFire's predefined named region attributes in these classes:

- org.apache.geode.cache.RegionShortcut. For peers and servers.
- org.apache.geode.cache.client.ClientRegionShortcut. For clients.

Shortcut attributes are a convenience only. They are just named attributes that Tanzu GemFire has already stored for you. You can override their settings by storing new attributes with the same id as the predefined attributes.

For a full list of all available region shortcuts, see Region Shortcuts Quick Reference.

The org.apache.geode.cache.RegionShortcut 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 Mode

- 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 Tanzu 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 Tanzu 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

This topic provides examples of storing and retrieving region shortcuts and custom named region attributes in VMware Tanzu GemFire.

Use these examples to get started with Tanzu GemFire region shortcuts.

Tanzu GemFire region shortcuts, in org.apache.geode.cache.RegionShortcut for peers and servers and org.apache.geode.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 Tanzu 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

This topic explains how to manage region attributes in VMware Tanzu GemFire.

Use region attributes to optimize 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>.

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>
  ```

  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
RegionFactory rf =
  cache.createRegionFactory(RegionShortcut.PARTITION);
rf.setPartitionResolver(new CustomerOrderResolver());
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");
```

- By issuing the gfsh create region command.

Modify Region Attributes

You can modify a region's event handlers and expiration and eviction attributes after the region is created.

> Do not modify attributes for existing regions unless absolutely necessary. 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.ComplicatedCacheListener</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());
```

- By issuing the gfsh alter region command. See alter region.

Creating Custom Attributes for Regions and Entries

This topic explains how create custom attributes for regions and entries in VMware Tanzu GemFire.

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.

> 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.

Building a New Region with Existing Content

This topic describes building a new region with existing content in VMware Tanzu GemFire.

A new region or cluster may need to be loaded with the data of an existing system. There are two approaches to accomplish this task. The approach used depends upon the organization of both the new and the existing cluster.

If both the number and the type of members is the same in both the new and the existing cluster, then the simplest option is to use backup and restore on the persistent disk store contents. Make a full online backup of the persistent data in the disk store of the existing cluster. Copy the files that comprise the backup to the new cluster location. A restore instills the data into the new cluster. See Creating Backups for System Recovery and Operational Management for details on how to make a backup and use the backup to restore a disk store.

Take a different approach when the number or the type of members is *not* the same in both the new and the existing cluster. This approach uses export and import of region data. Export the region data of the existing cluster to create a snapshot. Copy the snapshot to the new cluster location. Import the snapshot into the new cluster. See appropriate sections within Cache and Region Snapshots for details on making and using a snapshot.

Data Entries

This topic provides links to documentation about VMware Tanzu GemFire 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 VMware Tanzu GemFire requirements for data storage and distribution.

- Managing Data Entries

  Program your applications to create, modify, and manage your cached data entries.

- Copy on Read Behavior

  Set the copy-on-read region attribute to cause operations that get data to make a copy of the data, instead of returning a reference to the data.

- Requirements for Using Custom Classes in Data Caching

  Follow these guidelines to use custom domain classes for your cached entry keys and values.

Managing Data Entries

This topic describes how to program your applications to create, modify, and manage your cached data entries in VMware Tanzu GemFire.

> If you do not have the cache's `copy-on-read` attribute set to true, do not change the objects returned from the Java entry access methods. See Copy on Read Behavior.

Keys

Tanzu GemFire calls `hashCode()` on the key to map an entry within the region. The `hashCode()` return value must be the same for a given key on every server that hosts the region.

An `equals()` call return value on a given key also must be the same on every server that hosts the region.

A key may be a primitive type or a custom class. For custom classes, see Classes Used as Keys.

Do not use an enumerated type (`enum`) for a key. The `enum hashCode()` may not be overridden, and its hash code is based upon an address. Therefore, the return value for a `hashCode()` call can be different on each server, violating the restriction that it must return the same value on every server that hosts the region.

Create and Update Entries

To create or update an entry in the cache, use Region.put. For example:

```
String name = ...
String value = ...
this.currRegion.put(name,value);
```

> You can also use the gfsh put command to add entries to a region, and the get command to retrieve entries from a region. See get and put 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.

The getAll Operation

The batch operation Region.getAll takes a collection of keys and returns a Map of key-value pairs for the provided keys. If a given key does not exist in the region, then that key's value in the returned map will be null.

The putAll Operation

The batch operation Region.putAll takes a Map of key-value pairs, puts them into the cache, and then distributes them to all other members.

The design of a client application within a client-server design pattern faces the possibility that a partial operation can occur. Some, all, or none of the specified key-value pairs may be written with putAll. If either ServerOperationException or ServerConnectivityException is thrown, it can indicate an incomplete operation.

```
// Retry if the exception may be due to a transient cause.
for (int retry = 0; retry < 3; retry++) {
  throwable = null;
  try {
    region.putAll(map);
  } catch (ServerOperationException e) {
    throwable = e.getCause();
    if (!(e.getCause() instanceof TimeoutException ||
      e.getCause() instanceof LowMemoryException)) {
      // Not a transient exception. Do not retry.
      break;
    }
  } catch (ServerConnectivityException e) {
    throwable = e;
  }
}

if (throwable != null) {
  // Take appropriate action,
  // such as logging the exception and rethrowing it.
  System.out.println("putAll failed due to " + throwable);
  throw new Exception(throwable);
}
```

Either a ServerConnectivityException or a ServerOperationException with a cause of TimeoutException or LowMemoryException can indicate a transient error. A limited quantity of retries of putAll may result in a

completed operation. A repeated timeout may imply that the read-timeout value is not long enough to complete the bulk operation; use the org.apache.geode.cache.client.PoolFactory.setReadTimeout method to set the read-timeout value.

Client applications that cannot tolerate partial completion of a putAll operation may embed the operation into a transaction. See Transactions for details.

The processing of a map with many entries and/or extra-large data values may affect system performance and cause cache update timeouts, especially if the region uses overflow or persistence to disk. The processing may also cause a LowMemoryException to be thrown.

The removeAll Operation

The removeAll method takes a collection of keys and removes all of the entries for the specified keys from this region. This call performs the equivalent of calling destroy(Object) on this region once for each key in the specified collection. If an entry does not exist, then that key is skipped. An EntryNotFoundException is not thrown. This operation will be distributed to other caches if the region's scope is not set to Scope.LOCAL.

The processing of a map with many entries and/or extra-large data values may affect system performance and cause cache update timeouts, especially if the region uses overflow or persistence to disk. The processing may also cause a LowMemoryException to be thrown.

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.

Using gfsh to get and put

You can use the gfsh get and put commands to manage data. See get and put.

For example:

```
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
```

```
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
```

Copy on Read Behavior

This topic describes copy on read behavior in VMware Tanzu GemFire.

Methods that do a get-type of operation receive as a return value a direct reference to the cached object. This provides the value as quickly as possible, but it also makes possible code implementations that could incorrectly modify the referenced object, bypassing the distribution framework and causing region entries that are no longer consistent across cluster members.

The code that has the potential for harming cache consistency by using a reference to access and change a region entry is code that executes within the servers. Examples are cache writers and listeners, transactions, and functions. A client invocation of a get type of operation that is handled by the servers is not subject to this potential for harm, as the clients are in a distinct JVM from the servers, and references do not cross JVM boundaries. That client cannot receive a return value that is a direct reference to a region entry, as the servers hold the region entries and the servers do not reside within the client JVM.

To avoid modification of the referenced object, create a copy in one of two ways:

- Change the entry retrieval behavior for your cache by setting the copy-on-read cache attribute to true; its default value is false. When copy-on-read is true, all entry access methods return copies of the entries. This protects all server-side code from inadvertently modifying in-place. This attribute will negatively impact performance and memory consumption when a copy is not needed, as it takes time and memory to create the copy. Note that the copy-on-read attribute is applied at the cache level; it cannot be set for individual regions.

  There are two ways to set the copy-on-read attribute:

  - Set the attribute in the cache.xml file that defines the cache.

    ```
    <cache copy-on-read="true">
     ...
    </cache>
    ```

  - Use gfsh alter runtime to set the copy-on-read attribute once the servers have been started.

- Implement server-side code that creates and uses a copy of the returned object. For objects that are cloneable or serializable, copy the entry value to a new object using org.apache.geode.CopyHelper.copy. Example:

  ```
  Object o = region.get(key);
  StringBuilder s = (StringBuilder) CopyHelper.copy(o);
  // further operations on the region entry value will use s
  s.toUpperCase();
  ```

  Always use a Region method to then change data in the region. Do not use the reference returned from the entry access method. If the upper case string should become the new value for the region entry:

  ```
  region.put(key, s);
  ```

Requirements for Using Custom Classes in Data Caching

Follow these guidelines to use custom domain classes for your cached entry keys and values in VMware Tanzu GemFire.

Classloader

Each member's classloader must include classes for all objects the member accesses.

- For Java client applications, use the standard Java classpath parameters.
- For cache server and locator processes, use the gfsh start command's --classpath parameter. See Running GemFire Server Processes or Running GemFire Locator Processes.

Class Requirements

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's classloader 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

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 about the requirements and options for data serialization, see Data Serialization.

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.

Do not call hashCode() on an enum type data member within the key's custom hashCode() implementation. The enum hashCode() may not be overridden, and its hash is based upon an address. Therefore, an enumerated type's hashCode() return value can be different on each server, violating the restriction that hashCode() must return the same value on every server that hosts the region.

Topologies and Communication

This topic provides links to documentation that explains how to plan and configure VMware Tanzu GemFire member discovery, peer-to-peer, and client/server communication topologies.

- Topology and Communication General Concepts

  Before you configure your VMware Tanzu GemFire members, you should understand the options for topology and communication.

- Peer-to-Peer Configuration

  Use peer-to-peer configuration to set member discovery and communication within a single cluster.

- 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.

- Multi-site (WAN) Configuration

  Use the multi-site configuration to scale horizontally between disparate, loosely-coupled clusters. A wide-area network (WAN) is the main use case for the multi-site topology.

Topology and Communication General Concepts

This topic provides links to documentation about VMware Tanzu GemFire topology and general communication concepts.

Before you configure your VMware Tanzu GemFire members, you should understand the options for topology and communication.

- Topology Types

  The Tanzu GemFire topology options allow you to scale horizontally and vertically.

- Planning Topology and Communication

  Create a topology plan and a detailed list of machines and communication ports that your members will use. Configure your Tanzu GemFire systems and the communication between systems.

- How Member Discovery Works

  Tanzu GemFire provides various options for member discovery within a cluster and between clients and servers.

- How Communication Works

  Tanzu GemFire uses TCP/IP for communication between members. You can change the default settings to optimize communication for your system.

- 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 Tanzu GemFire across multiple cards. If no bind address setting is found, Tanzu GemFire uses the host machine's default address.

- Choosing Between IPv4 and IPv6

  By default, Tanzu GemFire uses Internet Protocol version 4 for Tanzu 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.

Topology Types

This topic describes topology types and options for VMware Tanzu GemFire.

The Tanzu GemFire topology options allow you to scale horizontally and vertically.

Tanzu GemFire provides a variety of cache topologies:

- At the core of all systems is the single, peer-to-peer cluster.
- 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 cluster is the building block for all Tanzu GemFire installations. Peer-to-peer alone is the simplest topology. Each cache instance, or member, directly communicates with every other member in the cluster. 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.

Cache instances in multiple application processes directly communicating with each other

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 Tanzu GemFire members. A client may also contain a local cache.

client/server topology described above

Multi-site Configuration

For horizontal scaling, you can use a loosely coupled multi-site topology. With multi-site, multiple Tanzu 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

This topic describes how to plan VMware Tanzu GemFire topology and communication.

Create a topology plan and a detailed list of machines and communication ports that your members will use. Configure your Tanzu GemFire systems and the communication between systems.

Determine Protocols and Addresses

Your configuration governs how your applications find each other and distribute events and data among themselves.

Work with your system administrator to determine the protocols and addresses you will use for membership and communication.

1. 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 and server.
2. Identify any members you want to run as standalone, isolated members with no member discovery. This can be a good option for clients, because it has faster startup but no peer-to-peer distribution of any kind.
3. For all non-standalone members:
    - Decide how many locators you will use and where they will run. To ensure the most stable startup and availability, use multiple locators 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.

Set Up Membership and Communication

Using the protocols and addresses you determined above, do the following:

- Set up membership in your systems.
- As needed, set up communication between your systems. See Configuring a Client/Server System.

How Member Discovery Works

This topic explains how VMware Tanzu GemFire member discovery works.

VMware Tanzu GemFire provides options for member discovery within a cluster and between clients and servers:

- Peer Member Discovery
- Standalone Member
- Client Discovery of Servers

Peer Member Discovery

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

Members discover each other using one or more locators. A locator provides both discovery and load balancing services. Peer locators manage a dynamic list of cluster members. New members connect to one of the locators to retrieve the member list, which it uses to join the system.

multiple peers, each with cache data, connected by locators

> Multiple locators ensure the most stable start up and availability for your cluster.

Standalone Member

The standalone member has no peers, does no peer discovery, and so does not use locators. It creates a cluster connection only to access the Tanzu 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.

> For performance and cache coherency, clients must run as standalone members or in different clusters 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.

multiple peers, each with cache data, connected by locators. Clients also connected to the same locators

Multi-site Discovery

In a multi-site (WAN) configuration, a Tanzu GemFire cluster uses locators to discover remote Tanzu GemFire clusters as well as to discover local Tanzu 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 Tanzu GemFire clusters to which it will connect for WAN distribution.

When a locator starts, 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 Tanzu GemFire clusters across the WAN.

For more information, see Discovery for Multi-Site Systems in *Overview of Multi-site Caching*.

How Communication Works

This topic explains how VMware Tanzu GemFire communication between members works.

Tanzu GemFire uses TCP/IP for communication between members. You can change the default settings 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, Tanzu GemFire uses TCP/IP for all communication.

How Multi-site (WAN) Systems Work

This topic explains how VMware Tanzu GemFire multi-site (WAN) systems work.

The Tanzu GemFire multi-site implementation connects distinct clusters. The clusters act as a single distributed system 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 cluster 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 clusters that are loosely coupled. Each site manages its own cluster, but region data is distributed to remote sites using one or more logical connections.

- Consistency for WAN Updates

  Tanzu GemFire ensures that all copies of a region eventually reach a consistent state on all members and clients that host the region, including Tanzu GemFire members that distribute region events across a WAN.

- Discovery for Multi-Site Systems

  Each Tanzu GemFire cluster in a WAN configuration uses locators to discover remote clusters as well as local members.

- Gateway Senders

A Tanzu GemFire cluster uses a *gateway sender* to distribute region events to another, remote Tanzu 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.

- Gateway Receivers

  A gateway receiver configures a physical connection for receiving region events from gateway senders in one or more remote Tanzu GemFire clusters.

Using Bind Addresses

This topic explains how to use bind addresses with VMware Tanzu GemFire.

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 Tanzu GemFire members to use non-default network cards on a multi-homed host.

> 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 about these specifications, see Choosing Between IPv4 and IPv6. Do not use host names for your address specifications. Host names resolve to default machine addresses.

Peer and Server Communication

You can configure peer, and server communication so that each communication type uses its own address or types use the same address. If no setting is found for a specific communication type, Tanzu GemFire uses the host machine's default address.

> 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 and server communication, ordered by precedence. For example, for server communication, Tanzu GemFire searches first for the cache-server bind address, then 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
cache.xml <cache-server> bind-address		X		<cache-server>bind-address=*address*

Property Setting Ordered by Precedence	Peer	Server	Gateway Receiver	Syntax
gfsh start server command-line --server-bind-address		X	X	gfsh start server --server-bind-address=*address*
gemfire.properties server-bind-address		X	X	server-bind-address=*address*
gemfire.properties bind-address	X	X	X	bind-address=*address*

For example, a member started with these configurations in its gemfire.properties and cache.xml files will use two separate addresses for peer and server communication:

```
// gemfire.properties setting for peer communication
bind-address=192.0.2.0
```

```
//cache.xml settings
<cache>

// Server communication
  <cache-server bind-address="192.0.2.1" ...

  <region> ...
```

Gateway Receiver Communication

If you are using multi-site (WAN) topology, you can also configure gateway receiver communication (in addition to peer and server communication) so that each communication type uses its own address.

This table lists the settings used for peer, server, and gateway receiver communication, ordered by precedence. For example, for gateway receiver communication, Tanzu GemFire searches first for a cache.xml <gateway-receiver> bind-address setting. If that is not set, Tanzu 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
cache.xml <gateway-receiver> bind-address			X	<gateway-receiver>bind-address=*address*
cache.xml <cache-server> bind-address		X		<cache-server>bind-address=*address*
gfsh start server command-line --server-bind-address		X	X	gfsh start server --server-bind-address=*address*
gemfire.properties server-bind-address		X	X	server-bind-address=*address*
gemfire.properties bind-address	X	X	X	bind-address=*address*

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=192.0.2.0
```

```
//cache.xml settings
<cache>
// Gateway receiver configuration
  <gateway-receiver start-port="1530" end-port="1551" bind-address="192.0.2.2"/>

// Server communication
  <cache-server bind-address="192.0.2.1" ...

  <region ...
```

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 Tanzu GemFire application, automatically start a co-located locator using the gemfire property start-locator, and specifying the bind address for it in that property setting.

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.

Choosing Between IPv4 and IPv6

This topic discusses choosing between IPv4 and IPv6 when using VMware Tanzu GemFire.

By default, Tanzu 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 should 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 Tanzu GemFire, IPv4 is generally recommended. IPv6 connections tend to take longer to form and the communication tends to be slower. Not all machines support IPv6 addressing. To use IPv6, all machines in your distributed system must support it or you will have connectivity problems.

> Do not mix IPv4 and IPv6 addresses. Use one or the other in all configurations.

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 Tanzu GemFire .

- IPv4:

```
192.0.2.0
```

- IPv6:

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

Peer-to-Peer Configuration

This topic explains how to configure VMware Tanzu GemFire peer-to-peer communication topology.

Use peer-to-peer configuration to set member discovery and communication within a single cluster.

- Configure Peer-to-Peer Discovery
- Organizing Peers into Logical Member Groups

Configure Peer-to-Peer Discovery

Peer members discover each other using one or more locators.

> Locator settings must be consistent throughout a cluster.

Allow Peer-to-Peer Discovery

To allow peer members to discover each other, add a list of your locators to the gemfire.properties file as a comma-separated list in the following format:

```
locators=LOCATOR-1-ADDRESS[PORT-1],LOCATOR-2-ADDRESS[PORT-2]
```

Where:

- LOCATOR-1-ADDRESS is the address of one of your locators.
- PORT-1 is the port number to use of the locator at this address.
- LOCATOR-2-ADDRESS is the address of another of your locators.
- PORT-2 is the port number to use of the locator at this address.

Example:

```
locators=mylocator1:8081,mylocator2:8082,mylocator2:8083
```

Disallow Peer-to-Peer Discovery

To run a standalone member, remove the value of the locators key in the gemfire.properties file ass follows:

```
locators=
```

Organize Peers into Logical Member Groups

You can use logical member groups to deploy JAR applications across multiple members or to execute functions across a member group.

1. To add a peer to a member group, do one of 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. To specify multiple member groups for a member, use a comma-separated list.
    - Use the gfsh command interface to start the member, providing the group name or group names as a parameter.

      Example gfsh command to start a server and associate it with member groups:

      ```
      gfsh>start server --name=server1 \
      --group=Portfolios,ManagementGroup1
      ```

      Example gfsh command to start a locator and associate it with member groups:

      ```
      gfsh>start locator --name=locator1 \
      --group=ManagementGroup1
      ```
2. After organizing peers into a member group, you can use the member group name to perform tasks such as deploy applications or execute functions.

   For example, run the following gfsh command to deploy an application across a member group:

   ```
   gfsh>deploy --jar=group1_functions.jar --group=ManagementGroup1
   ```

Client/Server Configuration

This topic provides links to documentation about VMware Tanzu GemFire client/server communication topologies.

In the client/server architecture, a relatively small server farm manages both the cached data and access to the cached 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.
- How Server Discovery Works

  VMware Tanzu 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.
- How Client/Server Connections Work

  The server pools in your VMware Tanzu 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.
- Configuring a Client/Server System

  Configure your server and client processes and data regions to run your client/server system.

- 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.

- Client/Server Example Configurations

  For easy configuration, you can start with these example client/server configurations and modify them for your systems.

- Fine-Tuning Your Client/Server Configuration

  You can fine-tune your client/server system with server load-balancing. 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 org.apache.geode.cache.server package.

Standard Client/Server Deployment

This topic described the most commonly used, standard VMware Tanzu GemFire client/server communication topology.

In this 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.

client/server data flow as described above

How Server Discovery Works

This topic explains how VMware Tanzu GemFire server discovery works.

Tanzu 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, Tanzu 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.

Locators track server availability and send clients to the most available servers. Clients send updates and requests to servers. Servers respond and send server events to clients.

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.

Servers split and clients split as described above.

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

This topic explains how VMware Tanzu GemFire client/server connections work.

The connection pools in your Tanzu 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, through pool connections and through subscription connections. Each kind of communication uses a different type of connection for maximum performance and availability.

Pool connections

Pool connections are 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. The figure below 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.

Pool connections for one Client and one Server. Server's listener for clients receives requests and from the Client and from other clients and respondes. Client connection pool sends requests to Server and from other servers and receives responses.

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 the pool's ping-interval.

If an application thread calls an operation that needs a connection and the pool cannot 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 closer attention as the client pool is able to connect only to servers at 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 a pool's open connection count over the pool's max-connections-per-server setting. If the max-connections-per-server 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 does not 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 does not 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.

Subscription connections

A subscription connection is used to stream cache 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.

Subscription connection that functions as described above added to Client

How the Pool Manages Subscription Connections

The pool's subscription connection is established 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.

The server sends ping messages once per second. You can adjust the interval with the system property gemfire.serverToClientPingPeriod, specified in milliseconds. The server sends its ping-interval setting to the client. The client then uses this and a multiplier to establish a read-timeout in the cache.

You can set the client property subscription-timeout-multiplier to enable timeout of the subscription feed with failover to another server.

Value options include:

- A value of zero (the default) deactivates timeouts.
- A value of one or more times-out the server connection after the specified number of ping intervals have elapsed. A value of one is not recommended. Set this to 2 or more to make sure the client will receive pings from the server before the timeout.

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 connection 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

This topic explains how to configure a VMware Tanzu GemFire client/server system. Use this topic to configure your server processes, 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 and Managing a Peer or Server Cache.
- Configure your clients as standalone applications. See Managing a Client Cache.
- Be familiar with cache region configuration. See Data Regions.
- Be familiar with server and client configuration properties. See cache.xml.

Procedure

1. 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"
      ```

2. 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="lucy1" port="41111"/>
       <locator host="lucy2" 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.

When TLS (SSL) is used clients can also be directed to go through a SNI gateway such as Istio or HAProxy to reach locators and servers. To do this add the following to your cache.xml pool configuration: <pool… > org.apache.geode.cache.client.proxy.SniProxySocketFactory my-gateway-address 12345

3. 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.

      > 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.

    - 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.
    - 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.

Organizing Servers Into Logical Member Groups

This topic explains how to organize VMware Tanzu GemFire servers into logical member groups.

In a client/server configuration, you can control which servers your clients use and target specific servers for specific data or tasks by putting servers into logical member groups. 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:

```
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

This topic provides example VMware Tanzu GemFire client/server configurations. You can start with these example client/server configurations and modify them for your systems.

Examples of Standard Client/Server Configuration

Generally, locators and servers use the same properties file, which lists locators as the discovery mechanism for peer members and for connecting clients. For example:

```
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. If you do not specify the member name, gfsh will automatically pick a random name. You can use this for automation.

The server's cache.xml declares a cache-server element, which identifies the JVM as a server in the cluster.

```
<cache>
  <cache-server port="40404" ... />
  <region . . .
```

Once the locator and server are started, the locator tracks the server as a peer in its cluster and as a server listening for client connections at port 40404.

You can also configure a cache server using the gfsh command-line utility. For example:

```
gfsh>start server --name=server1 --server-port=40404
```

See start server.

The client's cache.xml <client-cache> declaration automatically configures it as a standalone Tanzu 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 creating 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

This topic provides information about fine-tuning your VMware Tanzu GemFire client/server configuration.

You can fine-tune your client/server system with server load-balancing. 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 org.apache.geode.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 reduce the interval if you find your server loads fluctuating too much during normal operation. The lower you set this parameter, 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.

Tanzu 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 a server with a lower max-connections setting receives fewer connections than a server 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/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 org.apache.geode.cache.server package. The load for each server is weighted relative to the loads reported by other servers in the system.

Multi-site (WAN) Configuration

This topic provides links to documentation about VMware Tanzu GemFire multi-site (WAN) communication topologies.

Use the multi-site configuration to scale horizontally between disparate, loosely-coupled clusters.

The VMware Tanzu GemFire multi-site implementation connects disparate clusters. The systems act as a single system 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 cluster sites. A wide-area network (WAN) is the main use case for the multi-site topology.

For an overview, see How Multi-site (WAN) Systems Work.

For more detailed information, see the following topics:

- Multi-site (WAN) Topologies

  This topic explains the recommended VMware Tanzu GemFire multi-site topologies and the topologies to avoid.

- Configuring a Multi-site (WAN) System

  Plan and configure your multi-site topology, and configure the regions that will be shared between systems.

- Filtering Events for Multi-Site (WAN) Distribution

  Optionally, you can create gateway sender and 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.

- Resolving Conflicting Events

  Optionally, you can 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.

Overview of Multi-site Caching

This topic provides an overview of VMware Tanzu GemFire multi-site caching.

A multi-site installation consists of two or more loosely-coupled clusters. Each site manages its own cluster, 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 member caches. A site can use *serial* and/or *parallel* gateway sender configurations, as described in Gateway Senders.

Consistency for WAN Updates

Tanzu GemFire ensures that all copies of a region eventually reach a consistent state on all members and clients that host the region, including Tanzu 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 describes how Tanzu GemFire ensures consistency within a cluster, in client caches, and when applying updates over a WAN. Resolving Conflicting Events provides more details about implementing a custom conflict resolver for WAN updates.

Discovery for Multi-Site Systems

Each Tanzu GemFire cluster in a WAN configuration uses locators to discover remote clusters as well as local 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 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 Tanzu 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 Tanzu GemFire clusters.

Gateway Senders

A Tanzu GemFire cluster uses a *gateway sender* to distribute region events to another, remote Tanzu 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.

Tanzu GemFire provides two types of gateway sender configurations: *serial* gateway senders and *parallel* gateway senders.

Serial Gateway Senders

A *serial gateway sender* funnels region events through a single Tanzu GemFire server in the local cluster to a gateway receiver in the remote Tanzu 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 has a single distribution point, 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, so it may be a performance bottleneck. 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 from each of the Tanzu GemFire servers that host a partitioned region. For a partitioned region, each server that hosts primary buckets for the region uses its own logical queue to distribute events for those buckets. As you add new servers to scale the partitioned region, WAN distribution throughput scales automatically with each new instance of the parallel gateway sender.

Replicated regions cannot use a parallel gateway sender.

Although parallel gateway senders provide the best throughput for WAN distribution, they provide less control for event ordering. Event ordering for the region as a whole is not preserved, because multiple Tanzu GemFire servers distribute the regions events at the same time. However, the ordering of events for a given partition can be preserved. See Configuring Multi-Site (WAN) Event Queues.

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 Tanzu 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.

By default gateway sender queues use 5 threads to dispatch queued events. With a serial gateway sender, the single, logical queue that is hosted on a member is divided into multiple physical queues (5 by default) each with a dedicated dispatcher thread. You can configure whether the threads dispatch queued events by key, by thread, or in the same order in which events were added to the queue. For a parallel gateway sender, each logical queue that is hosted on a member is processed simultaneously by multiple threads.

See Configuring Multi-Site (WAN) Event Queues.

Timing of Connections

When a gateway sender is created, it tries once to connect to the gateway receiver. If the gateway sender cannot connect, the gateway sender does not retry until there is a batch of events to be sent. At that time, the gateway sender tries again to connect to the gateway receiver.

During a normal initial startup, one cluster site will start before the other cluster site. The cluster site that starts first will not be connected to the remote cluster site, even after that other cluster site starts. Only when data is to be sent will the connection be made.

High Availability for Gateway Senders

When a serial gateway sender configuration is deployed to multiple Tanzu 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. Tanzu 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, Tanzu GemFire ensures that the oldest running gateway sender operates as the primary.

A parallel gateway sender is deployed to multiple Tanzu GemFire members by default, and each member that hosts primary buckets for a partitioned region actively distributes data to the remote Tanzu 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 Tanzu GemFire member that hosts a redundant copy of those buckets takes over WAN distribution for those buckets.

Stopping Gateway Senders

The scope of the gateway sender stop operation is the VM on which it is invoked. When you stop a parallel gateway sender using the GatewaySender.stop() or gfsh stop gateway-sender, the gateway sender is stopped on the individual node where this API is called. If the gateway sender is not parallel (serial), then the gateway sender will stop on the local VM, and the secondary gateway sender will become primary and start dispatching events. The gateway sender will wait for GatewaySender.MAXIMUM_SHUTDOWN_WAIT_TIME seconds before stopping itself (by default, this value is set to 0). You can set this Java system property when starting the server member in gfsh. If the Java system property is set to -1, then the gateway sender process will wait until all events are dispatched from the queue before stopping.

> Use extreme caution when stopping parallel gateway senders by using the GatewaySender.stop() API or gfsh stop gateway-sender command.

The API and gfsh command stops the parallel gateway sender in one member, which causes data loss because events to buckets in that member will be dropped by the stopped sender. The partitioned region does not failover in this scenario since the member is still running. Instead, to ensure that the remaining events are sent, shut down the entire member to ensure proper failover of partition region events. When a member with the stopped parallel sender is shut down, the other parallel gateway sender members hosting the partition region become primary and deliver the remaining events. In addition, if the whole cluster is brought down after stopping an individual parallel gateway sender, then events queued on that gateway sender can be lost.

Pausing Gateway Senders

Similar to stopping a gateway sender, the scope of pausing a gateway sender is the VM on which it is invoked. Pausing a gateway sender temporarily stops the dispatching of events from the underlying queue. Note that events are still queued into the queue. In case where the gateway sender is parallel, the gateway sender is paused on the individual node where the GatewaySender.pause() API is called or the gfsh

pause gateway-sender command is invoked. The parallel gateway senders on other members can still dispatch events. In case where the paused gateway sender is not parallel (serial) and is not primary, then the primary gateway sender will still continue dispatching events. The batch of events that are in the process of being dispatched are dispatched regardless of the state of the pause operation. We can expect a maximum of one batch of events being received at the gateway receiver even after the gateway senders have been paused.

Gateway Receivers

A gateway receiver configures a physical connection for receiving region events from gateway senders in one or more remote Tanzu GemFire clusters.

A gateway receiver applies each region event to the same region or partition that is hosted in the local Tanzu 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 Tanzu GemFire members as needed for high availability and load balancing, however you can only host one gateway receiver per member.

After you create a gateway receiver, you can configure the gateway receiver to start automatically or to require a manual start. By default, the gateway receiver starts automatically (manual-start is set to false).

After you create and start a new gateway receiver at one WAN site, you can execute the load-balance gateway-sender command in gfsh for existing remote gateway senders so that the new receiver can pick up connections to gateway senders at different sites. You invoke this command on the gateway senders to redistribute connections more evenly among all the gateway receivers. Another option is to use the GatewaySender.rebalance Java API.

See Configure Gateway Receivers.

Multi-site (WAN) Topologies

This topic explains the recommended VMware Tanzu GemFire multi-site topologies and the topologies to avoid.

This section describes Tanzu 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 Tanzu 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.

Fully Connected Mesh Topology

A fully connected mesh network topology is one in which all sites know about each other. This is a robust configuration, as any one of the sites can go down without disrupting communication between the other sites. A fully connected mesh topology also guarantees that no site receives multiple copies of the same message.

A fully connected mesh with 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. This is likewise true for any other initiating site. If any site is removed, the remaining two are still fully connected.

Ring Topology

A ring topology is one in which each site forwards information to one other site, and the sites are connected in a circular manner. This figure shows a ring with three sites. In this topology, if site 1 sends updates to site 2, site 2 forwards the updates to site 3. No updates are forwarded to the original sender, so site 3 does not send the updates back to site 1.

A ring topology guarantees that every site receives one copy of each message sent by any site. In a ring, every site must stay up to maintain the connection. The failure of any site breaks the ability for updates to reach all sites. 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

There are numerous hybrid network topologies. Some of the sites are fully connected, while others form a ring.

The following figure shows a hybrid topology that forms a ring, with an extra connection that fully connects sites 1 and 3.

With this hybrid topology, if site 2 went down, it would not affect communication between sites 1 and 3. If site 3 went down, however, site 2 would not be able to send to site 1.

A second example hybrid topology is shown in the figure below. In this tree topology with site 1 as the root of the tree, sites 2 and 3 do not communicate with each another. This topology works for an application in which site 1 is a producer and the consumers (sites 2 and 3) have nothing to gain from being connected to each other. This topology also guarantees that no site receives the same update twice.

Unsupported Topologies

Topologies in which the same update may be delivered twice to a particular site do not work and are unsupported.

The DAG topology shown in this figure is an example of an unsupported technology. Site 4 will receive more than one copy of the same message when site 1 sends a message to sites 2 and 3, and sites 2 and 3 each forward the message to site 4.

Configuring a Multi-site (WAN) System

This topic explains how to plan and configure your VMware Tanzu GemFire multi-site topology, and how to configure the regions that are 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.

WAN deployments increase the messaging demands on a Tanzu GemFire system. To avoid hangs related to WAN messaging, always use the default setting of conserve-sockets=false for Tanzu GemFire members that participate in a WAN deployment. See Configuring Sockets in Multi-Site (WAN) Deployments and Making Sure You Have Enough Sockets.

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 for a description of different multi-site topologies.
2. Configure membership and communication for each cluster in your multi-site system. You must use locators for peer discovery in a WAN configuration. See Configuring Peer-to-Peer Discovery. Start each cluster using a unique distributed-system-id and identify remote clusters using remote-locators. For example:

   ```
   locators=<locator1-address>[<port1>],<locator2-address>[<port2>]
   distributed-system-id=1
   remote-locators=<remote-locator-addr1>[<port1>],<remote-locator-addr2>[<port2>]
   ```

3. Configure the gateway senders that you will use to distribute region events to remote systems. See Configure Gateway Senders.
4. Create the data regions that you want to participate in the multi-site system, specifying the gateway senders 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.

5. Configure gateway receivers in each Tanzu GemFire cluster that will receive region events from another cluster. See Configure Gateway Receivers.
6. Start cluster member processes in the correct order (locators first, followed by data nodes) to ensure efficient discovery of WAN resources. See Starting Up and Shutting Down Your System.
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.
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.
9. (Optional.) Configure persistence, conflation, and/or dispatcher threads for gateway sender queues using the instructions in Configuring Multi-Site (WAN) Event Queues.

Configure Gateway Senders

Each gateway sender configuration includes:

- 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 the amount of memory used by the queue, whether the queue is persisted to disk, and how one or more gateway sender threads dispatch events from the queue.

> To configure a gateway sender that uses gfsh to create the cache.xml configurations described below, as well as other options, see the information in create gateway-sender.

See WAN Configuration for more information about individual configuration properties.

1. For each Tanzu 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 Tanzu GemFire member that hosts a region that uses the sender. Regions using the same parallel gateway sender ID must be colocated.
   - You may choose to deploy a serial gateway sender configuration on one or more Tanzu 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 Tanzu GemFire member using gfsh, cache.xml, or Java API:
   - gfsh configuration command

     ```
     gfsh>create gateway-sender --id="sender2" --parallel=true --remote-distributed-system-id="2"

     gfsh>create gateway-sender --id="sender3" --parallel=true --remote-distributed-system-id="3"
     ```

   - cache.xml configuration

     These example cache.xml entries configure two parallel gateway senders to distribute region events to two remote Tanzu GemFire clusters (clusters “2” and “3”):

```
<cache>
  <gateway-sender id="sender2" parallel="true"
   remote-distributed-system-id="2"/>
  <gateway-sender id="sender3" parallel="true"
   remote-distributed-system-id="3"/>
   ...
</cache>
```

- Java configuration

  This example code shows how to configure a parallel gateway sender using the API:

```
// Create or obtain the 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:

```
gfsh>create gateway-sender --id=sender2 --parallel=true --remote-distributed-system-id=2 --maximum-queue-memory=150
```

     In cache.xml:

```
<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. For example:

```
gfsh>create gateway-sender --id=sender2 --parallel=true --remote-distributed-system-id=2 \
--maximum-queue-memory=150 --enable-persistence=true --disk-store-name=cluster2Store
```

     In cache.xml:

```
<gateway-sender id="sender2" parallel="true"
 remote-distributed-system-id="2"
 enable-persistence="true" disk-store-name="cluster2Store"
 maximum-queue-memory="150"/>
```

   - The number of dispatcher threads to use for processing events from each gateway queue. The dispatcher-threads attribute of the gateway sender specifies the number of threads that process the queue (default of 5). For example:

```
gfsh>create gateway-sender --id=sender2 --parallel=true --remote-distributed-system-id=2 \
--dispatcher-threads=2 --order-policy=partition
```

In cache.xml:

```
<gateway-sender id="sender2" parallel="false"
 remote-distributed-system-id="2"
 dispatcher-threads="2" order-policy="partition"/>
```

> When multiple dispatcher threads are configured for a serial queue, 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.

- For serial gateway senders (parallel=false) that use multiple dispatcher-threads, also configure the ordering policy to use for dispatching the events. See Configuring Dispatcher Threads and Order Policy for Event Distribution.
- Determine whether you should conflate events in the queue. See Conflating Events in a Queue.

> The gateway sender configuration for a specific sender id must be identical on each Tanzu GemFire member that hosts the gateway sender.

Create Data Regions for Multi-site Communication

When using a multi-site configuration, 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.

Note these important restrictions on the regions:

- Replicated regions cannot use a parallel gateway sender. Use a serial gateway sender instead.
- 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.
- Regions using the same parallel gateway sender ID must be colocated.
- gfsh Configuration

  ```
  gfsh>create region --name=customer --gateway-sender-id=sender2,sender3
  ```

  or to modify an existing region:

  ```
  gfsh>alter region --name=customer --gateway-sender-id=sender2,sender3
  ```

- 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");
```

> 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.

Configure Gateway Receivers

Always configure a gateway receiver in each Tanzu GemFire cluster that will receive and apply region events from another cluster.

A gateway receiver configuration can be applied to multiple Tanzu GemFire servers for load balancing and high availability. However, each Tanzu 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, Tanzu GemFire throws an exception. See Create Data Regions for Multi-site Communication.

> You can only host one gateway receiver per member.

A gateway receiver configuration specifies a range of possible port numbers on which to listen. The Tanzu 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.

You can optionally configure gateway receivers to provide a specific IP address or host name for gateway sender connections. If you configure hostname-for-senders, locators will use the provided host name or IP address when instructing gateway senders on how to connect to gateway receivers. If you provide "" or null as the value, by default the gateway receiver's bind-address will be sent to clients.

In addition, you can configure gateway receivers to start automatically or, by setting manual-start to true, to require a manual start. By default, gateway receivers start automatically.

> To configure a gateway receiver, you can use gfsh, cache.xml, or Java API configurations as described below. For more information about configuring gateway receivers in gfsh, see create gateway-receiver.

- gfsh configuration command

```
gfsh>create gateway-receiver --start-port=1530 --end-port=1551 \
    --hostname-for-senders=gateway1.example.com
```

- 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"
    hostname-for-senders="gateway1.example.com" />
  ...
</cache>
```

- Java API Configuration

```
// Create or obtain the cache
Cache cache = new CacheFactory().create();

// Configure and create the gateway receiver
GatewayReceiverFactory gateway = cache.createGatewayReceiverFactory();
gateway.setStartPort(1530);
gateway.setEndPort(1551);
gateway.setHostnameForSenders("gateway1.example.com");
GatewayReceiver receiver = gateway.create();
```

> When using the Java API, you must create any region that might receive events from a remote site before you create 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 yet exist. If you define regions in cache.xml, the correct startup order is handled automatically.

After starting new gateway receivers, you can execute the load-balance gateway-sender command in gfsh so that a specific gateway sender will be able to rebalance its connections and connect new remote gateway receivers. Invoking this command redistributes gateway sender connections more evenly among all the gateway receivers.

Another option is to use the GatewaySender.rebalance Java API.

As an example, assume the following scenario:

1. Create one receiver in site NY.
2. Create four senders in site LN.
3. Create three additional receivers in NY.

You can then execute the following in gfsh to see the effects of rebalancing:

```
gfsh -e "connect --locator=localhost[10331]" -e "list gateways"
...
(2) Executing - list gateways

GatewaySender Section

GatewaySender Id |            Member              | Remote Cluster Id |   Type   | Status  | Queued Events | Receiver Location
---------------- | ------------------------------- | ----------------- | -------- | ------- | ------------- | ------------------
ln               | mymac(ny-1:88641)<v2>:33491     | 2                 | Parallel | Running | 0             | mymac:5037
ln               | mymac(ny-2:88705)<v3>:29329     | 2                 | Parallel | Running | 0             | mymac:5064
ln               | mymac(ny-3:88715)<v4>:36808     | 2                 | Parallel | Running | 0             | mymac:5132
ln               | mymac(ny-4:88724)<v5>:52993     | 2                 | Parallel | Running | 0             | mymac:5324

GatewayReceiver Section

            Member              | Port | Sender Count | Senders Connected
------------------------------- | ---- | ------------ | ---------------------------------------------------------------------------------
mymac(ny-1:88641)<v2>:33491     | 5057 | 24           | ["mymac(ln-1:88651)<v2>:48277","mymac(ln-4:88681)<v5>:42784","mym
ac(ln-2:88662)<v3>:12796","mymac(ln-3:88672)<v4>:43675"]
mymac(ny-2:88705)<v3>:29329     | 5082 | 0            | []
mymac(ny-3:88715)<v4>:36808     | 5371 | 0            | []
mymac(ny-4:88724)<v5>:52993     | 5247 | 0            | []
```

Execute the load-balance command:

```
gfsh -e "connect --locator=localhost[10441]" -e "load-balance gateway-sender --id=ny"...

(2) Executing - load-balance gateway-sender --id=ny

            Member              | Result | Message
------------------------------- | ------ |---------------------------------------------------------------------------
mymac(ln-1:88651)<v2>:48277     | OK     | GatewaySender ny is rebalanced on member mymac(ln-1:88651)<v2>:48277
mymac(ln-4:88681)<v5>:42784     | OK     | GatewaySender ny is rebalanced on member mymac(ln-4:88681)<v5>:42784
mymac(ln-3:88672)<v4>:43675     | OK     | GatewaySender ny is rebalanced on member mymac(ln-3:88672)<v4>:43675
mymac(ln-2:88662)<v3>:12796     | OK     | GatewaySender ny is rebalanced on member mymac(ln-2:88662)<v3>:12796
```

Listing gateways in ny again shows the connections are spread much better among the receivers.

```
gfsh -e "connect --locator=localhost[10331]" -e "list gateways"...

(2) Executing - list gateways

GatewaySender Section

GatewaySender Id |            Member              | Remote Cluster Id |  Type    | Status  | Queued Events | Receiver Location
---------------- | ------------------------------- |  ---------------- | -------- | ------- | ------------- | ------------------
ln               | mymac(ny-1:88641)<v2>:33491      | 2                 | Parallel | Running | 0             | mymac:5037
ln               | mymac(ny-2:88705)<v3>:29329      | 2                 | Parallel | Running | 0             | mymac:5064
ln               | mymac(ny-3:88715)<v4>:36808      | 2                 | Parallel | Running | 0             | mymac:5132
ln               | mymac(ny-4:88724)<v5>:52993      | 2                 | Parallel | Running | 0             | mymac:5324

GatewayReceiver Section

        Member                  | Port | Sender Count | Senders Connected
--------------------------------- | ---- | ------------ | ------------------------------------------------------------------------------------------------------------------------------------
------------------------------
mymac(ny-1:88641)<v2>:33491       | 5057 | 9            |["mymac(ln-1:88651)<v2>:48277","mymac(ln-4:88681)<v5>:42784","myma
c(ln-3:88672)<v4>:43675","mymac(ln-2:88662)<v3>:12796"]
mymac(ny-2:88705)<v3>:29329       | 5082 | 4            |["mymac(ln-1:88651)<v2>:48277","mymac(ln-4:88681)<v5>:42784","myma
c(ln-3:88672)<v4>:43675"]
mymac(ny-3:88715)<v4>:36808       | 5371 | 4            |["mymac(ln-1:88651)<v2>:48277","mymac(ln-4:88681)<v5>:42784","myma
c(ln-3:88672)<v4>:43675"]
mymac(ny-4:88724)<v5>:52993       | 5247 | 3            |["mymac(ln-1:88651)<v2>:48277","mymac(ln-4:88681)<v5>:42784","myma
c(ln-3:88672)<v4>:43675"]
```

Running the load balance command in site ln again produces even better balance.

```
        Member                  | Port | Sender Count | Senders Connected
--------------------------------- | ---- | ------------ |------------------------------------------------------------------------------------------------------------------------------------
------------------------------
mymac(ny-1:88641)<v2>:33491       | 5057 | 7            |["mymac(ln-1:88651)<v2>:48277","mymac(ln-4:88681)<v5>:42784","myma
c(ln-2:88662)<v3>:12796","mymac(ln-3:88672)<v4>:43675"]
mymac(ny-2:88705)<v3>:29329       | 5082 | 3            |["mymac(ln-1:88651)<v2>:48277","mymac(ln-3:88672)<v4>:43675","myma
c(ln-2:88662)<v3>:12796"]
mymac(ny-3:88715)<v4>:36808       | 5371 | 5            |["mymac(ln-1:88651)<v2>:48277","mymac(ln-4:88681)<v5>:42784","myma
c(ln-2:88662)<v3>:12796","mymac(ln-3:88672)<v4>:43675"]
mymac(ny-4:88724)<v5>:52993       | 5247 | 6            |["mymac(ln-1:88651)<v2>:48277","mymac(ln-4:88681)<v5>:42784","myma
c(ln-2:88662)<v3>:12796","mymac(ln-3:88672)<v4>:43675"]
```

Accessing a Site Behind an SNI Proxy

Gateway senders can communicate with gateway receivers that are located on a network that provides access through an SNI proxy such as Envoy or HAProxy. To configure this, the following must be enabled or provided:

- WAN communication must be secured with TLS.
- The gateway receivers should be configured as described in Configuring One IP Address and Port to Access All Gateway Receivers in a Site. The value for hostname-for-senders and the associated port will be used to configure the SNI proxy.
- The locator, on the site behind the SNI proxy, must be started with the hostname-for-clients property. This value, and the associated port for the locator, will also be used to configure the SNI proxy.

The various ‘hostname-for-*’ values configured here will become the values placed into the TLS SNI extension for connections originating from the sending site. This allows the SNI proxy to correctly map requests to the correct backend servers.

The sending site must have its remote-locators property configured with a special SNI URI in the following form:

```
sni://sni_proxy_host:sni_proxy_port/locator_hostname_for_clients?dsid=N
```

Where:

- sni_proxy_host:sni_proxy_port is the address and port where the SNI proxy can be reached from the sending site.
- locator_hostname_for_clients is the value supplied to hostname-for-clients for the receiving site's locator.
- N is the distributed system ID of the receiving site.

The SNI proxy must be configured to map the various 'hostname-for-*' values, configured above, to the specific backend hosts and associated ports.

Filtering Events for Multi-Site (WAN) Distribution

This topic explains how to filter events for multi-site (WAN) distribution in VMware Tanzu GemFire.

You can create gateway sender and 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.

  Tanzu 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.

  Tanzu 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, Tanzu GemFire calls the listener's afterAcknowledgement method to indicate that is has received an ack from the remote site after the event was received.

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.

- XML example

```
<cache>
  <gateway-sender id="remoteA" parallel="true" remote-distributed-system-id="1">
    <gateway-event-filter>
      <class-name>org.apache.geode.util.SampleEventFilter</class-name>
      <parameter name="param1">
        <string>"value1"</string>
      </parameter>
    </gateway-event-filter>
  </gateway-sender>
</cache>
```

- gfsh example

```
gfsh>create gateway-sender --id=remoteA --parallel=true --remote-distributed-id="1"
--gateway-event-filter=org.apache.geode.util.SampleEventFilter
```

See create gateway-sender.

- API example

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

GatewayEventFilter efilter = new SampleEventFilter();

GatewaySenderFactory gateway = cache.createGatewaySenderFactory();
gateway.setParallel(true);
gateway.addGatewayEventFilter(efilter);
GatewaySender sender = gateway.create("remoteA", "1");
sender.start();
```

Resolving Conflicting Events

This topic explains how to resolve multi-site event conflicts in VMware Tanzu GemFire.

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.

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.

Implementing a GatewayConflictResolver

> 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.

Program the event handler

1. Create a class that implements the GatewayConflictResolver interface.
2. If you want to declare the handler in cache.xml, implement the org.apache.geode.cache.Declarable interface as well.
3. 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. For 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);
  }
}
```

> 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.

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

// Create and add a conflict resolver
cache.setGatewayConflictResolver(new MyConflictResolver);
```

Handling WAN Replication Errors

This topic discusses how to handle errors when using the VMware Tanzu GemFire WAN replication feature.

In Tanzu GemFire v10.2 and later, the REMOVE_FROM_QUEUE_ON_EXCEPTION property is deprecated and a new interface is introduced that enables you to access and manage previously inaccessible data. Additionally, in Tanzu GemFire v10.2 this data is stored in a customer-provided region.

The following solutions are available in v10.2:

1. A callback API
2. A gateway sender property
3. Two gateway receiver properties
4. Updated exception handling for exceptions from the gateway receiver
5. Storing gateway event contents in the user-provided region

Callback API

A new GatewayEventFailureListener callback API is defined as follows:

```
public interface GatewayEventFailureListener extends CacheCallback {
  /**
  * Callback invoked on the GatewaySender when an event fails to be processed by the
  * GatewayReceiver. The event will be removed from sender queue.
  *
  * @param event The event that failed
  *
  * @param exception The exception that occurred
  */
  void onFailure(GatewayQueueEvent<?, ?> event, Throwable exception);
}
```

The following example is a simple implementation that logs a warning when a callback is invoked.

```
public class LoggingGatewayEventFailureListener implements GatewayEventFailureListener {
  private Cache cache;

  @Override
  public void onFailure(GatewayQueueEvent event, Throwable exception) {
    this.cache.getLogger().warning("LoggingGatewayEventFailureListener onFailure: region="
        + event.getRegion().getName() + "; operation=" + event.getOperation() + "; key="
        + event.getKey() + "; value=" + event.getDeserializedValue() + "; exception=" + exception);
  }

  @Override
  public void initialize(Cache cache, Properties properties) {
    this.cache = cache;
  }
}
```

GatewaySender property

The GatewaySender can specify a GatewayEventFailureListener to handle failed events. This listener can be set using the GatewaySenderFactory, which also provides a method to retrieve it. The GatewayEventFailureListener property allows specifying a custom callback implementation or using the GemFire-provided solution with a user-provided region. If not specified, the behavior reverts to the default.

```
/**
* Sets the provided <code>GatewayEventFailureListener</code> in this GatewaySenderFactory.
*
* @param listener The <code>GatewayEventFailureListener</code>
*/
GatewaySenderFactory setGatewayEventFailureListener(GatewayEventFailureListener listener);
```

The GatewaySender adds the ability to get a GatewayEventFailureListener:

```
/**
* Returns this <code>GatewaySender's</code> <code>GatewayEventFailureListener</code>.
*
* @return this <code>GatewaySender's</code> <code>GatewayEventFailureListener</code>
*/
GatewayEventFailureListener getGatewayEventFailureListener();
```

Example to create gateway sender using the GemFire provided RegionGatewayEventFailureListener:

```
GatewaySender sender = cache.createGatewaySenderFactory()
   .setParallel(true)
   .setGatewayEventFailureListener(new RegionGatewayEventFailureListener(getCache(), "region_GatewayFailedEvents"))
   .create("ln", 2);
```

Gateway Sender gfsh Example

The following gfsh command creates a LoggingGatewayEventFailureLIstener:

```
gfsh > create gateway-sender --id=ln --remote-distributed-system-id=2 --gateway-event-failure-listener=LoggingGatewayEventFa
ilureListener

Member  | Status | Message
-------- | ------ | ----------------------------------------
server-1 | OK     | GatewaySender "ln" created on "server-1"
```

Gateway Sender cache.xml Example

```
<gateway-sender id="...">
<gateway-event-failure-listener>
   <class-name>LoggingGatewayEventFailureListener</class-name>
</gateway-event-failure-listener>
</gateway-sender>
```

GatewayReceiver Properties

The create gateway-receiver command defines these new arguments:

- event-retry-attempts - The number of retry attempts for failed events processed by the GatewayReceiver|
- event-retry-interval - The amount of time to wait between retry attempts for failed events processed by the GatewayReceiver XML Configuration gateway-sender|

The GatewayReceiverFactory has the following enhancements:

- Adds the ability to set the number of retry attempts (setEventRetryAttempts).
- Adds the ability to set the wait time between retry attempts (setEventRetryInterval).

```
/**
* Sets the number of retry attempts to apply failing events from remote GatewaySenders.
* When an event is being retried, processing of other events will be blocked until a successful
* retry, or until the number of retry attempts has been reached.
*
* After retrying all failing events from remote GatewaySenders, if there are still failing
* events, those events will be sent back to GatewaySenders and the GatewayEventFailureListener
* callbacks will be invoked on senders.
*
* Sets retry attempts to -1, if want to keep retrying until the event is processed successfully.
*
* @param eventRetryAttempts The retry attempts for the event
* @return this factory
* @since GemFire 10.2
*/
GatewayReceiverFactory setEventRetryAttempts(int eventRetryAttempts);

/**
* Sets the wait time between retry attempts to apply failing events from remote GatewaySenders
*
* @param eventRetryInterval The wait time in milliseconds between retries
* @return this factory
* @since GemFire 10.2
*/
GatewayReceiverFactory setEventRetryInterval(long eventRetryInterval);
```

The GatewayReceiver adds the ability to get event retry attempts and event retry interval:

```
/**
* Returns the number of times to retry a failing event before throwing an exception.
*
* @return the number of times to retry a failing event before throwing an exception
* @since GemFire 10.2
*/
int getEventRetryAttempts();

/**
* Returns the amount of time in milliseconds to wait between attempts to apply a failing event.
*
* @return the amount of time in milliseconds to wait between attempts to apply a failing event
* @since GemFire 10.2
*/
long getEventRetryInterval();
```

Example to create gateway receiver setting those new properties:

```
GatewayReceiver receiver = cache.createGatewayReceiverFactory()
   .setEventRetryAttempts(1)
   .setEventRetryInterval(100)
   .create();
```

Gateway Receiver gfsh Example

The following gfsh command creates a gateway receiver which sets retry related properties:

```
gfsh>create gateway-receiver --event-retry-attempts=2 --event-retry-interval=6000

Command result:
 Member  | Status | Message
-------- | ------ | ---------------------------------------------------------------------------------
server-7 | OK     | GatewayReceiver created on member "server-7" and will listen on the port "5357"
```

gateway receiver cache.xml Example

```
<gateway-receiver event-retry-attempts="1" event-retry-interval="100">

</gateway-receiver>
```

Updated exception handling for exceptions from the gateway receiver

Exceptions that occur from the gateway receiver site are treated differently to maintain data consistency:

1. For certain exceptions like LowMemoryException and PartitionOfflineException, GemFire will continue retrying the same event on the receiving site until the critical memory or disk issue is solved. There is no benefit to using the new API to remove the failed event from the queue, as these exceptions will continue occurring on the receiver site until the issue is addressed. Events will be retried indefinitely on the receiver site and the LowMemoryException and PartitionOfflineException will not be sent to sender side.
2. For other types of exceptions, once retry attempts are exhausted, the sender will use the new callback API if one exists.

The following examples show how to use the GemFire provided region implementation of the GatewayEventFailureListener:

Using java API: This example creates a gateway sender that sets the RegionGatewayEventFailureListener with the provided RegionName.

```
String regionName = "region_GatewayFailedEvents";
RegionGatewayEventFailureListener listener = new RegionGatewayEventFailureListener(getCache(), regionName);
GatewaySenderFactory sender = getCache().createGatewaySenderFactory();
sender.setGatewayEventFailureListener(listener);
return sender.create(senderId, 2);
```

Using gfsh command: This example creates a gateway sender that sets the RegionGatewayEventFailureListener with the provided RegionName.

```
gfsh > create gateway-sender \
--id=ln --remote-distributed-system-id=2 \
--gateway-event-failure-listener=org.apache.geode.cache.wan.RegionGatewayEventFailureListener{'regionName':'region_Gatew
ayFailedEvents'}
```

Using cache.xml: This example creates a gateway sender which sets the RegionGatewayEventFailureListener with the provided RegionName.

```
<gateway-sender id="ln" remote-distributed-system-id="2">
  <gateway-event-failure-listener>
      <class-name>org.apache.geode.cache.wan.RegionGatewayEventFailureListener</class-name>
      <parameter name="regionName">
          <string>region_GatewayFailedEvents</string>
      </parameter>
  </gateway-event-failure-listener>
</gateway-sender>
```

Storing gateway event contents in the user-provided region

When the onFailure method of the RegionGatewayEventFailureListener is invoked, it will put failed gateway events in the user defined region in the GatewaySender using the following key, value pair: (RegionGatewayEventFailureListener.FailedEventRegionKey RegionGatewayEventFailureListener.FailedEventRegionValue

- RegionGatewayEventFailureListener.FailedEventRegionKey has the following fields:

```
private DistributedMember member;
private String senderId;
private int remoteDSId;
private long counter;
```

- RegionGatewayEventFailureListener.FailedEventRegionValue has the following fields:

```
private String regionPath;
private Object key;
private Object value;
private Operation operation;
private Object callbackArgument;
private Throwable exception;
private long failedTime;
private boolean possibleDuplicate;
```

> On the GemFire provided RegionGatewayEventFailureListener, the exception thrown from the gateway receiver site is saved to the region supplied. This helps in debugging the cause of the event failure on the receiver site. However, this can also put memory pressure on the cluster on the sender site as it takes extra memory to store the exceptions. Consider overflowing the region to disk, or use your own implementation of the listener to store only exception message instead of the whole exception object.

```
public class ExampleRegionGatewayEventFailureListener
    implements GatewayEventFailureListener, GatewaySenderAware {
  private volatile Cache cache;
  private volatile String regionName;
  private volatile String senderId;
  private volatile int remoteDSId;
  private final AtomicLong counter = new AtomicLong();

  private volatile Region<FailedEventRegionKey, FailedEventRegionValue> region;
  private volatile boolean regionDestroyed;

  private static final String regionNotExistMessage =
      "Region {} for storing failed gateway events for sender {} was not found";
  private static final String regionExistMessage =
      "Region {} to store failed gateway events is found for sender {}";

  public static final String REGION_NAME_PROPERTY = "regionName";

  private static final Logger logger = LogService.getLogger();

  public ExampleRegionGatewayEventFailureListener() {}

  public ExampleRegionGatewayEventFailureListener(Cache cache, String regionName) {
    this.cache = cache;
    this.regionName = regionName;
  }

  public void setRegionName(String regionName) {
    this.regionName = regionName;
  }

  @Override
  public void setSenderId(String senderId) {
    this.senderId = senderId;
  }

  @Override
  public void setRemoteDSId(int remoteDSId) {
    this.remoteDSId = remoteDSId;
  }

  @Override
  public void onFailure(GatewayQueueEvent<?, ?> event, Throwable exception) {
    GatewaySenderEventImpl gatewaySenderEvent = (GatewaySenderEventImpl) event;

    if (region == null) {
      try {
        region = cache.getRegion(regionName);
        if (region == null) {
          return;
        }
        logger.info(regionExistMessage, regionName, senderId);
      } catch (Exception ignore) {
        // still can not find the region to store failed gateway events;
        return;
      }
    }

    FailedEventRegionKey key =
        new FailedEventRegionKey(cache.getDistributedSystem().getDistributedMember(), senderId,
            remoteDSId, counter.incrementAndGet());
    FailedEventRegionValue value = getValue(gatewaySenderEvent, exception);
```

```
  try {
    region.put(key, value);
    if (regionDestroyed) {
      regionDestroyed = false;
      logger.info("Region {} for storing failed gateway events for sender {} has been recreated",
          regionName, senderId);
    }
  } catch (RegionDestroyedException regionDestroyedException) {
    if (!regionDestroyed) {
      logger.warn("Region {} for storing failed gateway events for sender {} has been destroyed",
          regionName, senderId, regionDestroyedException);
      regionDestroyed = true;
    }
  } catch (Exception e) {
    if (logger.isDebugEnabled()) {
      logger.debug("Can not store failed gateway event {} for sender {}", event, senderId, e);
    }
  }
}

private FailedEventRegionValue getValue(GatewaySenderEventImpl event, Throwable exception) {
  String regionPath = event.getRegion().getFullPath();
  Object key = event.getKey();
  Object value = null;
  Operation operation = event.getOperation();
  Object callbackArgument = event.getCallbackArgument();
  boolean possibleDuplicate = event.getPossibleDuplicate();
  try {
    value = event.getDeserializedValue();
  } catch (Exception ignore) {
  }
  return new FailedEventRegionValue(regionPath, key, value, operation, callbackArgument,
      exception, System.currentTimeMillis(), possibleDuplicate);
}

@Override
public void initialize(Cache cache, Properties properties) {
  this.cache = cache;
  if (properties == null) {
    return;
  }
  regionName = properties.getProperty(REGION_NAME_PROPERTY);

  try {
    region = cache.getRegion(regionName);
    if (region == null) {
      logger.warn(regionNotExistMessage, regionName, senderId);
    } else {
      logger.info(regionExistMessage, regionName, senderId);
    }
  } catch (Exception e) {
    logger.warn(regionNotExistMessage, regionName, senderId, e);
  }
}

public static class FailedEventRegionKey implements DataSerializable {
  private static final long serialVersionUID = 1527713216173281L;

  private DistributedMember member;
  private String senderId;
  private int remoteDSId;
  private long counter;

  public FailedEventRegionKey() {}
```

```
FailedEventRegionKey(DistributedMember member, String senderId, int remoteDSId, long counter) {
  this.member = member;
  this.senderId = senderId;
  this.remoteDSId = remoteDSId;
  this.counter = counter;
}

public String getSenderId() {
  return senderId;
}

public DistributedMember getMember() {
  return member;
}

public int getRemoteDSId() {
  return remoteDSId;
}

@Override
public void toData(DataOutput out) throws IOException {
  writeObject(member, out);
  writeString(senderId, out);
  writeInteger(remoteDSId, out);
  writeLong(counter, out);
}

@Override
public void fromData(DataInput in) throws IOException, ClassNotFoundException {
  member = DataSerializer.readObject(in);
  senderId = readString(in);
  remoteDSId = readInteger(in);
  counter = readLong(in);
}

@Override
public boolean equals(Object o) {
  if (this == o) {
    return true;
  }

  if (!(o instanceof FailedEventRegionKey)) {
    return false;
  }

  FailedEventRegionKey regionKey = (FailedEventRegionKey) o;
  return regionKey.remoteDSId == remoteDSId
      && regionKey.counter == counter
      && Objects.equals(regionKey.member, member)
      && Objects.equals(regionKey.senderId, senderId);
}

@Override
public int hashCode() {
  int result = 1;

  result = member == null ? result : 31 * result + member.hashCode();
  result = senderId == null ? result : 31 * result + senderId.hashCode();
  result = 31 * result + remoteDSId;
  result = 31 * result + Long.hashCode(counter);

  return result;
}
```

```
  @Override
  public String toString() {
    StringBuilder builder = new StringBuilder();
    builder.append(getClass().getSimpleName()).append("@").append(Integer.toHexString(hashCode()))
        .append("(")
        .append("member=").append(member)
        .append("; senderId=").append(senderId)
        .append("; remoteDSId=").append(remoteDSId)
        .append("; counter=").append(counter)
        .append(")");
    return builder.toString();
  }
}

public static class FailedEventRegionValue implements DataSerializable {
  private static final long serialVersionUID = 313845192130669535L;

  private String regionPath;
  private Object key;
  private Object value;
  private Operation operation;
  private Object callbackArgument;
  private Throwable exception;
  private long failedTime;
  private boolean possibleDuplicate;

  public FailedEventRegionValue() {}

  FailedEventRegionValue(String regionPath, Object key, Object value, Operation operation,
      Object callbackArgument, Throwable exception, long failedTime, boolean possibleDuplicate) {
    this.regionPath = regionPath;
    this.key = key;
    this.value = value;
    this.operation = operation;
    this.callbackArgument = callbackArgument;
    this.exception = exception;
    this.failedTime = failedTime;
    this.possibleDuplicate = possibleDuplicate;
  }

  public String getRegionPath() {
    return regionPath;
  }

  public Object getKey() {
    return key;
  }

  public Object getValue() {
    return value;
  }

  public Operation getOperation() {
    return operation;
  }

  public Object getCallbackArgument() {
    return callbackArgument;
  }

  public Throwable getException() {
    return exception;
  }
```

```
public long getFailedTime() {
  return failedTime;
}

public boolean isPossibleDuplicate() {
  return possibleDuplicate;
}

@Override
public void toData(DataOutput out) throws IOException {
  writeString(regionPath, out);
  writeObject(key, out);
  writeObject(value, out);
  writeObject(operation, out);
  writeObject(callbackArgument, out);
  writeObject(exception, out);
  writeLong(failedTime, out);
  writeBoolean(possibleDuplicate, out);
}

@Override
public void fromData(DataInput in) throws IOException, ClassNotFoundException {
  regionPath = readString(in);
  key = readObject(in);
  value = readObject(in);
  operation = readObject(in);
  callbackArgument = readObject(in);
  exception = readObject(in);
  failedTime = readLong(in);
  possibleDuplicate = readBoolean(in);
}

@Override
public boolean equals(Object o) {
  if (this == o) {
    return true;
  }

  if (!(o instanceof RegionGatewayEventFailureListener.FailedEventRegionValue)) {
    return false;
  }

  FailedEventRegionValue regionValue = (FailedEventRegionValue) o;
  return regionValue.failedTime == failedTime
      && regionValue.possibleDuplicate == possibleDuplicate
      && Objects.equals(regionValue.regionPath, regionPath)
      && Objects.equals(regionValue.key, key)
      && Objects.equals(regionValue.value, value)
      && Objects.equals(regionValue.operation, operation)
      && Objects.equals(regionValue.callbackArgument, callbackArgument)
      && Objects.equals(regionValue.exception, exception);
}

@Override
public int hashCode() {
  int result = 1;

  result = regionPath == null ? result : 31 * result + regionPath.hashCode();
  result = key == null ? result : 31 * result + key.hashCode();
  result = value == null ? result : 31 * result + value.hashCode();
  result = operation == null ? result : 31 * result + operation.hashCode();
  result = callbackArgument == null ? result : 31 * result + callbackArgument.hashCode();
  result = exception == null ? result : 31 * result + exception.hashCode();
```

```
    result = 31 * result + Long.hashCode(failedTime);
    result = 31 * result + Boolean.hashCode(possibleDuplicate);

    return result;
  }

  @Override
  public String toString() {
    StringBuilder builder = new StringBuilder();
    builder.append(getClass().getSimpleName()).append("@").append(Integer.toHexString(hashCode()))
        .append("(")
        .append("regionPath=").append(regionPath)
        .append(", key=").append(key)
        .append(", value=").append(value)
        .append(", operation=").append(operation)
        .append(", callbackArgument=").append(callbackArgument)
        .append(", exception=").append(exception)
        .append(", failedTime=").append((new DateTime(failedTime)))
        .append(", possibleDuplicate=").append(possibleDuplicate)
        .append(")");
    return builder.toString();
  }
 }
}
```

Managing Tanzu GemFire

This topic describes how to plan and implement tasks associated with managing, monitoring, and troubleshooting Tanzu GemFire.

- Tanzu GemFire Management and Monitoring

  Tanzu GemFire provides APIs and tools for managing your distributed system and monitoring the health of your distributed system members.

- Managing Heap and Off-heap Memory

  By default, Tanzu GemFire uses the JVM heap. Tanzu GemFire also offers an option to store data off heap. This section describes how to manage heap and off-heap memory to best support your application.

- Disk Storage

  With Tanzu 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.

- 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 into another, such as capturing data from a production system and moving it into a smaller QA or development system.

- Region Compression

  This section describes region compression, its benefits and usage.

- Network Partitioning

  Tanzu GemFire architecture and management features help detect and resolve network partition problems.

- Security

  The security framework establishes trust by authenticating components and members upon connection. It facilitates the authorization of operations.

- Performance Tuning and Configuration

  A collection of tools and controls allow you to monitor and adjust Tanzu GemFire performance.

- Logging

  Comprehensive logging messages help you confirm system configuration and debug problems in configuration and code.

- Statistics

  Every application and server in a distributed system can access statistical data about Tanzu GemFire operations. You can configure the gathering of statistics by using the `alter runtime` command of `gfsh` or in the `gemfire.properties` file to facilitate system analysis and troubleshooting.

- Troubleshooting and System Recovery

  This section provides strategies for handling common errors and failure situations.

Tanzu GemFire Management and Monitoring

This topic provides links to documentation for the VMware Tanzu GemFire APIs and tools for managing your cluster and monitoring the health of your members.

- Management and Monitoring Features

  VMware Tanzu GemFire uses a federated Open MBean strategy to manage and monitor all members of the cluster. This strategy gives you a consolidated, single-agent view of the cluster.

- Overview of Tanzu GemFire Management and Monitoring Tools

  Tanzu GemFire provides a variety of management tools you can use to manage a Tanzu GemFire cluster.

- Architecture and Components

  Tanzu 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 cluster. All members in the cluster are manageable through MBeans and Tanzu GemFire Management Service APIs.

- JMX Manager Operations

  Any member can host an embedded JMX Manager, which provides a federated view of all MBeans for the cluster. 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.

- Federated MBean Architecture

  Tanzu GemFire uses MBeans to manage and monitor different parts of Tanzu GemFire. Tanzu GemFire's federated MBean architecture is scalable and allows you to have a single-agent view of a Tanzu GemFire cluster.

- Configuring RMI Registry Ports and RMI Connectors

  Tanzu GemFire programmatically emulates out-of-the-box JMX provided by Java and creates a JMXServiceURL with RMI Registry and RMI Connector ports on all manageable members.

- Executing gfsh Commands through the Management API

  You can also use management APIs to execute gfsh commands programmatically.

Management and Monitoring Features

This topic explains the management and monitoring features in VMware Tanzu GemFire.

Tanzu GemFire uses a federated Open MBean strategy to manage and monitor all members of the cluster. This strategy gives you a consolidated, single-agent view of the cluster.

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 Tanzu GemFire administration architecture:

- Tanzu GemFire monitoring is tightly integrated into Tanzu GemFire's processes instead of running in a separately installed and configured monitoring agent. You can use the same framework to actually manage Tanzu GemFire and perform administrative operations, not just monitor it.
- All Tanzu GemFire MBeans are *MXBeans*. They represent useful and relevant information about the state of the cluster and all its members. Because MXBeans use the Open MBean model with a predefined set of types, clients and remote management programs do not require access to

model-specific classes representing your MBean types. Using MXBeans adds flexibility to your selection of clients and makes the Tanzu GemFire management and monitoring much easier to use.

- Each member in the cluster is manageable through MXBeans, and each member hosts its own MXBeans in a Platform MBeanServer.
- Any Tanzu GemFire member can be configured to provide a federated view of all the MXBeans for all members in a Tanzu GemFire cluster.
- Tanzu 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 cluster by using any third-party tool that is compliant with JMX. For example, JConsole.

References

For more information about MXBeans and Open MBeans, see:

- Annotation Type MXBean on the Oracle documentation website.
- Package javax.management.openmbean on the Oracle documentation website.

Overview of GemFire Management and Monitoring Tools

This topic provides an overview of VMware Tanzu GemFire management and monitoring tools, and links to topics that provide additional details about these tools.

The Tanzu GemFire management and monitoring tools allow you to configure all members and processes of a cluster, monitor operations in the system, and start and stop the members.

Internally, Tanzu GemFire uses Java MBeans, specifically MXBeans, to expose management controls and monitoring features. You can monitor and control Tanzu GemFire by creating and using Java programs that use these MXBeans, or you can use one of several tools provided with Tanzu GemFire to monitor and manage your cluster. The primary tool for these tasks is the gfsh command-line tool, as described in this section.

Tanzu GemFire provides the following tools to manage a Tanzu GemFire installation:

gfsh Command-line tool

The gfsh command line tool provides a set of commands you use to configure, manage, and monitor a cluster. gfsh is the recommended tool for managing your cluster.

Use gfsh to:

- Start and stop Tanzu GemFire processes, such as locators and cache servers
- Deploy applications
- Create and destroy regions
- Execute functions
- Manage disk stores
- Import and export data
- Monitor Tanzu GemFire processes
- Launch Tanzu GemFire monitoring tools
- Shut down a cluster

- Script various operations involving Tanzu GemFire members
- Save the configuration for all members of a cluster

gfsh runs in its own shell, or you can execute gfsh commands directly from the OS command line. gfsh can interact with remote systems using the http protocol. You can also write scripts that run in a gfsh shell to automate system startup.

You can use gfsh to create shared cluster configurations for your cluster. You can define configurations that apply to the entire cluster, or that apply only to groups of similar members that all share a common configuration. Tanzu GemFire locators maintain these configurations as a hidden region and distribute the configuration to all locators in the cluster. The locator also persists the shared configurations on disk as cluster.xml and cluster.properties files. You can use those shared cluster configuration files to restart your system, migrate the system to a new environment, add new members to a cluster, or to restore existing members after a failure.

A basic cluster configuration consists of:

- cluster.xml file shared by the cluster
- cluster.properties file shared by the cluster
- Deployed jar files containing application Java classes.

See Overview of the Cluster Configuration Service and Cluster Configuration Files and Troubleshooting for additional details about gfsh cluster configuration files.

Using the gfsh tool, you can easily migrate a Tanzu GemFire-based application from a development environment into a testing or production environment.

Executing gfsh commands with the management API

You can also use Tanzu GemFire's management APIs to execute gfsh commands in a Java class. See Executing gfsh Commands through the Management API.

Member Configuration Management

When you issue gfsh commands and have the cluster configuration service enabled (on a locator), Tanzu GemFire saves the configurations created within gfsh by building a cluster.xml and cluster.properties files for the entire cluster, or group of members.

You can also directly create configurations using cache.xml and gemfire.properties files and manage the members individually.

Java Management Extension (JMX) MBeans

Tanzu GemFire uses a federated Open MBean strategy to manage and monitor all members of the cluster. Your Java classes interact with a single MBeanServer that aggregates MBeans from other local and remote members. Using this strategy gives you a consolidated, single-agent view of the cluster.

Tanzu GemFire's implementation of JMX is industry-standard and friendly to generic JMX clients. You can monitor or manage the cluster by using any third-party tool that is compliant with JMX. For example, JConsole.

See VMware Tanzu GemFire Management and Monitoring

Tanzu GemFire Java API

The Tanzu GemFire API provides a set of Java classes you can use to manage and monitor a cluster. See the org.apache.geode.management package in the javadocs.

Tanzu GemFire Management Console

The Tanzu GemFire Management Console offers a user-friendly graphical interface to streamline operations, diagnostics, and management of GemFire deployments. Key capabilities include:

- Centralized view of multiple clusters
- Real-time cluster throughput monitoring
- Region exploration and creation within clusters
- Simplified JAR file deployment and removal
- Disk store management and creation
- Cluster log search and review
- Web-based access to the GemFire Shell (gfsh) command line
- Function deployment and execution
- Gateway Sender and Receiver management

For detailed instructions on downloading, installing, and using the Tanzu GemFire Management Console, visit the Tanzu GemFire Management Console documentation.

Tanzu GemFire Pulse

Tanzu GemFire Pulse is a Web Application that provides a graphical dashboard for monitoring vital, real-time health and performance of Tanzu GemFire clusters, members, and regions.

> GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

Use Pulse to examine total memory, CPU, and disk space used by members, uptime statistics, client connections, and critical notifications. Pulse communicates with a Tanzu GemFire JMX manager to provide a complete view of your Tanzu GemFire deployment.

See Tanzu GemFire Pulse.

JConsole

JConsole is a JMX monitoring utility provided with a Java Development Kit (JDK). You use gfsh to connect to Tanzu GemFire, and then launch JConsole with a gfsh command. The JConsole application allows you to browse MBeans, attributes, operations, and notifications. For more information, see Browsing Tanzu GemFire MBeans through JConsole.

Prometheus Metrics Endpoint

Tanzu GemFire provides a Prometheus endpoint that allows the metrics to be gathered by a metrics collection agent and forwarded to a metrics monitoring platform, such as the Tanzu GemFire Management Console, for further analysis and alerting.

For more information, see Prometheus Metrics Endpoint.

Tanzu Observability

Tanzu GemFire provides out-of-the-box integration with Tanzu Observability. For more information, see Tanzu Observability.

Prometheus Metrics Endpoint

This topic explains how to configure the Prometheus metrics endpoint provided by VMware Tanzu GemFire.

Tanzu GemFire provides a Prometheus endpoint that allows the metrics to be gathered by a metrics collection agent and forwarded to a metrics monitoring platform (such as the VMware Tanzu GemFire Management Console) for further analysis and alerting.

Configure Tanzu GemFire Metrics

To enable the metrics endpoints across your Tanzu GemFire cluster, the HTTP service must be enabled on each member.

- On locators, enable it with the enable-management-rest-service property (default: true).
- On servers, enable it with the --start-rest-api option (default: false). The metrics endpoint runs on the port specified by http-service-port property.

Once the HTTP service is enabled, configure the emission setting for each member to control which metrics are emitted:

- Default (the default): emits a curated subset of ~250 metrics.
- All: emits all available metrics.
- None: disables metric emission.

Property	Variable	Accepted Values and Example	Default
emission	--J=-Dgemfire.prometheus.metrics.emission=*value*	• Default: Emit approximately 200 metrics. • All: Emit all GemFire metrics. • None: No metrics will be available on the endpoint. Example: --J=-Dgemfire.prometheus.metrics.emission=All	*No Default Set*

interval	--J=-Dgemfire.prometheus.metrics.interval=*value*	A positive integer followed by a unit of time. • s: Seconds • m: Minutes • d: Days • w: Weeks Examples: • --J=-Dgemfire.prometheus.metrics.interval=4m • --J=-Dgemfire.prometheus.metrics.interval=90s	1m (1 minute) : If emission is set to All, the default becomes 2m.

Configure GemFire

To configure metrics for a member (a GemFire locator or server), you may provide the following information when you create the member:

- HTTP Service Port: Specify a unique HTTP service port mapping for each locator and server using the --http-service-port=8001 option in the gfsh start command.
- Metrics Emission: Specify the amount of metrics data for each locator and server. In your gfsh start command, use the --J=-Dgemfire.prometheus.metrics.emission option to specify the emission level and its value. For example, --J=-Dgemfire.prometheus.metrics.emission=All. If not set, it will default to Default.

Example Startup

1. Start a locator and specify the metrics emission level.

```
start locator --name=locator1 --http-service-port=8001 --J=-Dgemfire.prometheus.metrics.emission=Default
```

2. After the locator has started, locate and record the IP address of the locator in the output of the start command. For example, in the sample output below, the IP address is 192.168.50.245:

```
Starting a GemFire Locator in /Users/username/my_gemfire/locator1...
.................
Locator in /Users/username/my_gemfire/locator1 on 192.168.50.245[10334] as locator1 is currently online.
Process ID: 4580
Uptime: 10 seconds
GemFire Version: 10.1.0
Java Version: 11.0.17
Log File: /Users/username/my_gemfire/locator1/locator1.log
JVM Arguments: -Dgemfire.enable-cluster-configuration=true
-Dgemfire.load-cluster-configuration-from-dir=false
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path:
//Users/username/my_gemfire/locator1/gemfire-bootstrap-10.0.1.jar

Successfully connected to: JMX Manager [host=192.168.50.245, port=1099]

Cluster configuration service is up and running.
```

3. Verify that the metrics module is properly configured by visiting the following URL:

```
http://HOSTNAME:8001/metrics
```

Where HOSTNAME is the IP address of the locator that you recorded in the previous step.

For example, using the example information above, the URL would be http://192.168.50.245:8001/metrics.

The output should resemble the following:

```
# HELP gemfire_replyWaitTime
# TYPE gemfire_replyWaitTime gauge
gemfire_replyWaitTime{category="DistributionStats",instance="distributionStats",member="192.168.
129.137(locator1:76435:locator)<ec><v0>:41000",} 0.0
# HELP gemfire_loadsCompleted
# TYPE gemfire_loadsCompleted gauge
gemfire_loadsCompleted{category="CachePerfStats",instance="RegionStats-managementRegionStats",member="19
2.168.129.137(locator1:76435:locator)<ec><v0>:41000",} 0.0
gemfire_loadsCompleted{category="CachePerfStats",instance="cachePerfStats",member="192.168.129.137(locator1:
76435:locator)<ec><v0>:41000",} 0.0
...
```

4. Now, start a server, setting the metrics emission level as well as specifying --start-rest-api.

```
start server --name=server1 --http-service-port=8002 --start-rest-api --J=-Dgemfire.prometheus.metrics.emission=Defa
ult
```

5. After the server has started, locate and record the IP address of the server in the output of the start command.
6. Verify that the metrics module is properly configured by visiting the following URL:

```
http://HOSTNAME:8002/metrics
```

Where HOSTNAME is the IP address of the server that you recorded in the previous step. The metrics output should look similar to the output in step 3.

7. Repeat these steps for each additional member of the cluster, changing the HTTP service port for each member.

Security

Tanzu GemFire supports the ability to configure authentication and authorization. If you configure security on your members, you will need to provide credentials when accessing the metrics endpoint. Accessing the metrics endpoint will require the permission CLUSTER:READ.

Tanzu GemFire also provides support for configuring SSL/TLS, which is supported by the metrics endpoint. If you have SSL configured, you will need to use the relevant certificates when accessing the metrics endpoint.

For more information, consult the documentation on configuring Tanzu GemFire security.

Tanzu Observability

This topic discusses how to configure Tanzu Observability with VMware Tanzu GemFire and provides an example.

Tanzu GemFire's Prometheus Metrics Endpoint provides built-in integration with Tanzu Observability. Prometheus adds a metrics endpoint to a specified Tanzu GemFire member.

By default, this endpoint hosts approximately 200 Tanzu GemFireGemFire metrics at an update interval of one minute. These metrics can be scraped by a metrics collection agent (such as Telegraf) and forwarded to a metrics monitoring platform (such as Tanzu Observability) for further analysis and alerting.

Delivering Tanzu GemFire Metrics through Telegraf to Tanzu Observability

To enable metrics across your Tanzu GemFire cluster see the Prometheus Metrics Endpoint documentation for starting and configuring the Tanzu GemFire cluster.

After the Tanzu GemFire cluster is configured you must configure Telegraf and the Tanzu Observability Proxy.

Telegraf and Tanzu Observability Proxy Setup Example

This example uses Telegraf as the agent to pull Prometheus-style metrics from a Tanzu GemFire cluster. Telegraf sends the metrics to a local proxy, which forwards them to the Tanzu Observability service.

Install Telegraf and the Proxy

1. Install Telegraf and the Proxy for your operating system, using the instructions found in the product documentation:
    - Linux
    - MacOS
    - Windows
2. Acquire an API token that authenticates communication from the Proxy by following the instructions in Generating an API Token in the product documentation.

Configure Telegraf

1. After installation, add the following inputs.prometheus section to your Telegraf configuration file with your selected tags and the URL to your metrics endpoints. Your Telegraf configuration file should contain the specified tags and URLs to instruct Telegraf to scrape metrics from GemFire's Prometheus metrics endpoints.

    Example configuration file named "tanzu-gemfire.conf":

```
# Telegraf config to scrape GemFire metrics
[agent]
  interval = "10s"
[[inputs.prometheus]]
  urls = ["http://localhost:7001/metrics","http://localhost:8001/metrics"]
  # These tags are used in the Tanzu Observability-GemFire integration. Set them to uniquely identify your GemFire clu
ster.
  [inputs.prometheus.tags]
    "label.gemfire-environment" = "milky-way"
    "label.gemfire-cluster" = "my-cluster"
[[outputs.wavefront]]
    host = "localhost"
    port = 2878
    metric_separator = "."
    source_override = ["hostname", "agent_host", "node_host"]
    convert_paths = true
    use_regex = false
```

2. Restart Telegraf to use the new configuration:

   Linux:

   ```
   sudo service telegraf restart
   ```

   MacOS:

   ```
   brew services restart telegraf
   ```

   Windows:

   ```
   net stop telegraf
   net start telegraf
   ```

3. In a browser, navigate to your GemFire dashboard in Tanzu Observability. You should see live metrics. To find your Tanzu Observability dashboard, see instructions in Tanzu Observability.

4. You can also view the logfile in a shell window:

   ```
   tail -f /usr/local/var/log/wavefront/wavefront.log
   ```

   The output should resemble the following:

   ```
   2021-06-02 11:59:20,210 INFO  [proxy:checkin] Checking in: https://vmware.wavefront.com/api
   2021-06-02 11:59:20,210 INFO  [proxy:checkin] Checking in: https://vmware.wavefront.com/api
   2021-06-02 11:59:29,915 INFO  [AbstractReportableEntityHandler:printStats] [2878] Points received rate: 302 pps (1
   min), 296 pps (5 min), 1588 pps (current).
   2021-06-02 11:59:29,915 INFO  [AbstractReportableEntityHandler:printStats] [2878] Points received rate: 302 pps (1
   min), 296 pps (5 min), 1588 pps (current).
   2021-06-02 11:59:29,915 INFO  [AbstractReportableEntityHandler:printStats] [2878] Points delivered rate: 295 pps (1
   min), 294 pps (5 min)
   2021-06-02 11:59:29,915 INFO  [AbstractReportableEntityHandler:printStats] [2878] Points delivered rate: 295 pps (1
   min), 294 pps (5 min)
   ```

Verification and Troubleshooting

If everything is working properly, your cluster should be listed within the Tanzu GemFire integration in Tanzu Observability.

If everything is not working properly:

- GemFire
  - Verify that the metrics endpoint is hosting metrics by curling one of the metrics endpoints or viewing it in your browser. For example: curl localhost:7001/metrics
  - View the member's log and verify that the metrics module is loaded.
- Telegraf
  - View the Telegraf logs or start Telegraf in console mode to catch any suppressed errors.
  - Ensure that the scraped URLs contain the correct metrics ports.
- Proxy
  - Verify that the Proxy logs are actively receiving datapoints.
  - Verify that the Proxy has the correct subdomain, <your-subdomain>.wavefront.com, and a valid API key.

Default Dashboard Metrics

These tables list the metrics available in the default Tanzu Observability dashboard.

Tanzu GemFire Cluster Metrics

Metric	Description
Uptime	Maximum uptime of all members in this cluster
Server Count	Count of members that are servers and serve up data in this cluster
Locator Count	Count of locators in this cluster. The locator is a Tanzu GemFire process that tells new, connecting members where running members are located and provides load balancing for server use.
Region Count	Count of regions in this Tanzu GemFire cluster. Regions hold entries (key/value pairs) in buckets.
CPU Count	The current count of CPUs recognized by Tanzu GemFire
Cluster Entries	This metric represents the last value registered for the total entry count across all regions and data in the specified Tanzu GemFire cluster. For partitioned data, we take a raw sum of the region entries. For replicate data, we only count the primary copy and only use the result from one server.
Server Old Gen Utilization	For each server in this cluster, this chart shows the amount of memory, in bytes, being used by each server and the overall percentage of that member's available memory usage on the right axis.
Disk Utilization	Disk utilization by member in the cluster. The percentage remaining is computed by using the statistics gathered by Tanzu GemFire from the underlying system. Some operating systems (like macOS) do not emit these metrics which will cause this graph to be empty.
CPU Utilization % by Member	The current utilization percentage of this cluster's CPUs
Current Client Connects by Member	This chart displays a line for each server in the cluster and represents the number of external clients connected to that member.
Sampler Delay Duration	Delay duration measures the actual time (in ms) the Tanzu GemFire statistics sampler slept. It calculates the difference between when it went to sleep and when it woke up to sample. Sample time shows how long it took to collect the sample after it woke up.

Metric	Description
Cluster Communication Delays	Tanzu GemFire uses replyWaitsInProgress as a means to measure intra-cluster communication and determine a stalled or failing member.
IO Waits	The time spent waiting to read or write.
Abandoned Reads/Second	Tanzu GemFire monitors the AbandonedReadRequests from the cache server.
CPU Steal Time	Steal time is the percentage of time a virtual CPU waits for a real CPU while the hypervisor is servicing another virtual processor. Your virtual machine shares resources with other instances on a single host in a virtualized environment. One of the resources it shares is CPU Cycles.

Tanzu GemFire Features Metrics

Metric	Description
Gets Throughput	This chart displays the throughput of all Get operations performed across the entire cache. RIGHT Axis (RED): The Get operation requests by the remote Client applications. LEFT Axis (BLUE): The Get operation requests by the remote Client applications and distributed get operations.
Average Get Latency	Average Get Latency presents the total time taken by all Get operations performed across the cluster divided by the number of all Get operations performed across the cluster to get an average time taken per Get operation in the system.
Cache Hits Per Second	The rate of Get operations across the entire cluster that resulted in a matched key per second.
Cache Misses Per Second	This chart represents the rate of Get operations that resulted in a miss, where the key that was requested was not in the cache per second.
Puts Throughput	This chart displays the throughput of all Put operations performed across the entire cache. RIGHT Axis (RED): The Put operation requests by the remote Client applications. LEFT Axis (BLUE): The Put operation requests by the remote Client applications and distributed Put operations
Average Put Latency	Average Put Latency presents the total time taken by all Put operations performed across the cluster divided by the number of all Put operations performed across the cluster to get an average time taken per Put operation in the system.
Cache Hit Ratio	The Cache hit ratio is currently based on system Get operations and any Get operation across the cluster. It represents the percentage of those Get operations that return a value based on the specified key.
Function Execution Details	RED (RIGHT AXIS): Function Execution Queue size. BLUE (LEFT AXIS): A calculated rate based on functionExecutionCalls minus functionExecutionsCompleted, which provides the output of FAILED function execution calls per second.
Client Put Request By Server	For each server member, this chart represents the rate of requests of Put operations received per second on a specific member.
Persistent Region Overcapacity	For persistent regions that hold data in memory and on disk, we provide this chart to show whether the capacity of system is sufficient to accommodate the stored data.

Metric	Description
Message Queue Size	Size of the Tanzu GemFire message queue in number of operations (such as Get, Put, and Destroy) that it contains
Client Query Rate	The number of queries received per second from GemFire clients
Average Query Time	The average query time is computed by taking the total time of all query requests (processQueryTime) and dividing it by the number of requests (queryRequests), to get an average time spent per query.
Region Details	This table shows the type, name, and entry count for each region in the cache. For convenience, we have summarized the Partitioned regions with the Partition Total rows.

WAN Gateway Metrics

Metric	Description
WAN Receiver Throughput	This chart shows all of the Gateway Receivers for this cluster and the rate of bytes/second that are sent (red) or received (blue).
WAN Sender Queue	This graph displays the rate (messages/second) at which a Gateway Sender is able to send events to a WAN connected cluster.

Architecture and Components

This topic explains VMware Tanzu GemFire architecture and components.

Tanzu GemFire's management and monitoring system consists of a single JMX Manager node and one or more managed nodes within a cluster. All members in the cluster are manageable through MBeans and Tanzu GemFire Management Service APIs.

Architecture

The following diagram depicts the architecture of the management and monitoring system components.

In this architecture every Tanzu GemFire member is manageable. All Tanzu GemFire MBeans for the local Tanzu 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 cluster 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:

- Local MXBeans that represent the locally monitored components on the node. See List of Tanzu GemFire JMX MBeans for a list of possible MXBeans existing for the managed node.
- Built-in platform MBeans.

JMX Manager Node

A JMX Manager node is a member that can manage other Tanzu GemFire members—that is, other managed nodes—as well as itself. A JMX Manager node can manage all other members in the cluster.

To convert a managed node to a JMX Manager node, you configure the Tanzu 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 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 cluster.
- Local MXBeans that represent the locally monitored components on this node, same as any other managed node.
- Aggregate MXBeans:
  - DistributedSystemMXBean
  - DistributedRegionMXBean
  - DistributedLockServiceMXBean
- ManagerMXBean with Scope=ALL, which allows various cluster-wide operations.
- Proxy to MXBeans on managed nodes.
- Built-in platform MXBeans.

JMX Integration

Management and monitoring tools such as gfsh command-line interface and Pulse use JMX/RMI as the communication layer to connect to Tanzu GemFire nodes. All Tanzu 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.

> GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

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 Tanzu 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 Tanzu 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. Tanzu GemFire manages the inter-JVM communication required to provide a federated view of all MBeans in the cluster.

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 about how to configure the RMI registry and RMI connector, see Configuring RMI Registry Ports and RMI Connectors.

Management APIs

Tanzu GemFire management APIs represent the Tanzu 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 Tanzu GemFire monitoring and management.

The entry point to Tanzu 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 cluster. The implementation of getManagementService is a singleton for now but may eventually support multiple cache instances.

You can use the Tanzu 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 Tanzu 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 org.apache.geode.management package for more details.

You can also execute gfsh commands using the ManagementService API. See Executing gfsh Commands through the Management API and the JavaDocs for the org.apache.geode.management.cli package.

Tanzu GemFire Management and Monitoring Tools

This section lists the currently available tools for managing and monitoring Tanzu GemFire:

- gfsh: VMware Tanzu GemFire command-line interface that provides a simple and powerful command shell that supports the administration, debugging and deployment of Tanzu GemFire applications. It features context sensitive help, scripting and the ability to invoke any commands from within the application using a simple API. See gfsh.
- Tanzu GemFire Pulse: Easy-to-use, browser-based dashboard for monitoring Tanzu GemFire deployments. Tanzu GemFire Pulse provides an integrated view of all Tanzu GemFire members within a cluster. See Tanzu GemFire Pulse.
- Pulse Data Browser: This Tanzu GemFire Pulse utility provides a graphical interface for performing OQL ad-hoc queries in a Tanzu GemFire cluster. See Data Browser.
- JConsole: A JMX-based management and monitoring tool provided in the Java 2 Platform that provides information about the performance and consumption of resources by Java applications. See Using JConsole in the Java SE documentation on the Oracle website.
- Java VisualVM (jvisualvm): 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 about using jvisualvm, see jvisualvm in the Java SE documentation on the Oracle website.

Starting a JMX Manager

This topic explains how to start a JMX manager in VMware Tanzu GemFire.

JMX Manager nodes are members that manage themselves and other Tanzu GemFire members. A JMX Manager node can manage all other members in the cluster. Typically, a locator functions as the JMX Manager, but you can also convert any other member (such as a server) into a JMX Manager node.

To allow a server to become a JMX Manager, you configure Tanzu GemFire property jmx-manager=true in the server's gemfire.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 cluster, the server node is started as a JMX Manager node.

To force a server to become a JMX Manager node whenever it is started, set the Tanzu 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 cluster, 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 cluster. However, you can run more than one 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. If you do not want a locator to become a JMX manager, then you must use the jmx-manager=false property when you start the locator.

> GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

After the node becomes a JMX Manager, all other jmx-manager-* configuration properties listed in Configuring a JMX Manager 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 Tanzu GemFire Locator in /Users/username/apache-geode/locator1...
....
Locator in /Users/username/apache-geode/locator1 on 192.0.2.0[10334] as locator1
is currently online.
Process ID: 27144
Uptime: 5 seconds
Tanzu GemFire Version: 10.2
Java Version: 1..0_361
Log File: /Users/username/apache-geode/locator1/locator1.log
JVM Arguments: -Dgemfire.enable-cluster-configuration=true
-Dgemfire.load-cluster-configuration-from-dir=false
-Dgemfire.launcher.registerSignalHandlers=true
-Djava.awt.headless=true -Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /Users/username/apache-geode/lib/gemfire-core-1.2.0.jar
:/Users/username/apache-geode/lib/gemfire-dependencies.jar

Successfully connected to: JMX Manager [host=192.0.2.0, port=1099]

Cluster configuration service is up and running.
```

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.

Configuring a JMX Manager

In the gemfire.properties file, you configure a JMX manager as follows.

Property	Description	Default
http-service-port	If non-zero, then Tanzu GemFire starts an embedded HTTP service that listens on this port. The HTTP service is used to host the Tanzu GemFire Pulse Web application. If you are hosting the Pulse web app on your own Web server, then deactivate this embedded HTTP service by setting this property to zero. Ignored if jmx-manager is false.	7070
http-service-bind-address	If set, then the Tanzu GemFire member binds the embedded HTTP service to the specified address. If this property is not set but the HTTP service is enabled using http-service-port, then Tanzu GemFire binds the HTTP service to the member's local address.	*not set*
jmx-manager	If true then this member can become a JMX Manager. All other jmx-manager-* properties are used when it does become a JMX Manager. If this property is false then all other jmx-manager-* properties are ignored. The default value is true on locators.	false (with Locator exception)
jmx-manager-access-file	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 com.sun.management.jmxremote.access.file system property. Ignored if jmx-manager is false or if jmx-manager-port is zero.	*not set*
jmx-manager-bind-address	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 jmx-manager is false or jmx-manager-port is zero. This address also applies to the Tanzu GemFire Pulse server if you are hosting a Pulse web application.	*not set*

Property	Description	Default
jmx-manager-hostname-for-clients	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 jmx-manager is false or if jmx-manager-port is zero.	*not set*
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.	*not set*
jmx-manager-port	Port on which this JMX Manager listens for client connections. If this property is set to zero, Tanzu 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-enabled	If true and jmx-manager-port 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 Tanzu GemFire connections. Ignored if jmx-manager 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 jmx-manager 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 statistic-sample-rate. Setting this value too high causes gfsh and Tanzu 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 --dir=locator1
Stopping Locator running in /home/user/test2/locator1 on ubuntu.local[10334] as locator1...
Process ID: 2081
Log File: /home/user/test2/locator1/locator1.log
....
No longer connected to ubuntu.local[1099].
```

For a server:

```
gfsh>stop server --dir=server1
Stopping Cache Server running in /home/user/test2/server1 ubuntu.local[40404] as server1...
Process ID: 1156
Log File: /home/user/test2/server1/server1.log
....


No longer connected to ubuntu.local[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.

JMX Manager Operations

Any member can host an embedded JMX Manager, which provides a federated view of all MBeans for the VMware Tanzu GemFire cluster. 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 must have a JMX Manager started in your Tanzu GemFirecluster to use Tanzu GemFire management and monitoring tools such as gfsh and Tanzu GemFire Pulse.

To create MBeans, a Security Manager must be enabled. See Enable Security with Property Definitions for more information.

> Each node that acts as the JMX Manager has additional memory requirements depending on the number of resources that it is managing and monitoring. Being a JMX Manager can increase the memory footprint of any process, including locator processes. See Memory Requirements for Cached Data for more information on calculating memory overhead on your Tanzu GemFire processes.

- Starting a JMX Manager
- Configuring a JMX Manager
- Stopping a JMX Manager

Federated MBean Architecture

This topic describes VMware Tanzu GemFire's federated MBean architecture.

Tanzu GemFire uses MBeans to manage and monitor different parts of Tanzu GemFire. Tanzu GemFire's federated MBean architecture is scalable and allows you to have a single-agent view of a Tanzu GemFire cluster.

Federation of Tanzu 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 cluster.

Tanzu GemFire federation takes care of the following functionality:

- MBean proxy creation
- MBean state propagation
- Notifications propagation
- Operation invocation

MBean Proxy Naming Conventions

Each Tanzu 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, Tanzu 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

Tanzu GemFire proxies are inherently local MBeans. Every Tanzu 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.

> Aggregate MBeans on the JMX Manager node are not proxied.

List of GemFire JMX MBeans

This topic provides descriptions for the management and monitoring MBeans that are available in VMware Tanzu GemFire, and links to documentation that provides more information about MBeans.

The following diagram illustrates the relationship between the different JMX MBeans that have been developed to manage and monitor VMware Tanzu GemFire.

Diagram illustrates the relationship between the different JMX MBeans that have been developed to manage and monitor GemFire

- JMX Manager MBeans

  This section describes the MBeans that are available on the JMX Manager node.

- Managed Node MBeans

  This section describes the MBeans that are available on all managed nodes.

JMX Manager MBeans

This topic describes the VMware Tanzu GemFire MBeans available on the JMX notifications.

The JMX Manager node includes all local beans listed under Managed Node MBeans and the following beans that are available only on the JMX Manager node:

- ManagerMXBean
- DistributedSystemMXBean
- DistributedRegionMXBean
- DistributedLockServiceMXBean

ManagerMXBean

Represents the Tanzu 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.

> You must configure the node to allow it to become a JMX Manager. See Configuring a JMX Manager 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 org.apache.geode.management.ManagerMXBean JavaDocs for information about available MBean methods and attributes.

DistributedSystemMXBean

System-wide aggregate MBean that provides a high-level view of the entire cluster including all members (cache servers, peers, locators) and their caches. At any given point of time, it can provide a snapshot of the complete cluster and its operations.

The DistributedSystemMXBean provides APIs for performing cluster-wide operations such as backing up all members, shutting down all members or showing various cluster metrics.

You can attach a standard JMX NotificationListener to this MBean to listen for notifications throughout the cluster. See Tanzu GemFire JMX MBean Notifications 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 Tanzu GemFire System.

MBean Details

Scope	Aggregate
Proxied	No
Object Name	GemFire:type=Distributed,service=System
Instances Per Node	1

See the org.apache.geode.management.DistributedSystemMXBean JavaDocs for information about 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 org.apache.geode.management.DistributedRegionMXBean JavaDocs for information about 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 cluster defined by a specified distribution manager. Any number of DistributedLockService instances can be created with different service names. For all processes in the cluster 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=LockService,name=<dlsName>
Instances Per Node	0..N

See the org.apache.geode.management.DistributedLockServiceMXBean JavaDocs for information about 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
- CacheServerMXBean
- RegionMXBean
- LockServiceMXBean
- DiskStoreMXBean
- AsyncEventQueueMXBean
- LocatorMXBean
- LuceneServiceMXBean

JMX Manager nodes will have managed node MBeans for themselves since they are also manageable entities in the cluster.

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 Tanzu GemFire resources. Any JMX client can connect to the MBean server and start managing a Tanzu GemFire Member by using this MBean.

See MemberMXBean Notifications 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 org.apache.geode.management.MemberMXBean JavaDocs for information about available MBean methods and attributes.

CacheServerMXBean

Represents the Tanzu GemFire CacheServer. Provides data and notifications about server, subscriptions, durable queues and indices.

See CacheServerMXBean Notifications for a list of notifications emitted by this MBean.

MBean Details

Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=CacheServer,member=<name-or-dist-member-id>
Instances Per Node	1

See the org.apache.geode.management.CacheServerMXBean JavaDocs for information about available MBean methods and attributes.

RegionMXBean

Member's local view of region.

MBean Details

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 org.apache.geode.management.RegionMXBean JavaDocs for information about available MBean methods and attributes.

LockServiceMXBean

Represents a named instance of a LockService . Any number of LockServices can be created in a member.

MBean Details

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 org.apache.geode.management.LockServiceMXBean JavaDocs for information about available MBean methods and attributes.

DiskStoreMXBean

Represents a DiskStore object which provides disk storage for one or more regions

MBean Details

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 org.apache.geode.management.DiskStoreMXBean JavaDocs for information about available MBean methods and attributes.

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,queue=<queue-id>,member=<name-or-dist-member-id>
Instances Per Node	0..N

See the org.apache.geode.management.AsyncEventQueueMXBean JavaDocs for information about 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 org.apache.geode.management.LocatorMXBean JavaDocs for information about available MBean methods and attributes.

LuceneServiceMXBean

The member's local view of existing Lucene indexes.

MBean Details

Scope	Local
Proxied	Yes

Object Name	GemFire:service=CacheService,name=LuceneService,type=Member,member=<name-or-dist-member-id>
Instances Per Node	0..1

See the org.apache.geode.cache.lucene.management.LuceneServiceMXBean JavaDocs for information about available MBean methods and attributes.

GatewaySenderMXBean

A GatewaySenderMXBean represents a gateway sender.

MBean Details

Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=GatewaySender,gatewaySender=<sender-id>,member=<name-or-dist-member-id>
Instances Per Node	0..1

See the org.apache.geode.management.GatewaySenderMXBean JavaDocs for information about available MBean methods and attributes.

GatewayReceiverMXBean

A GatewayReceiverMXBean represents a gateway receiver.

MBean Details

Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=GatewayReceiver,member=<name-or-dist-member-id>
Instances Per Node	0..1

See the org.apache.geode.management.GatewayReceiverMXBean JavaDocs for information about available MBean methods and attributes.

Browsing GemFire MBeans through JConsole

This topic explains how to browse all the VMware Tanzu GemFire MBeans in your cluster using JConsole.

To view Tanzu GemFire MBeans through JConsole, perform the following steps:

1. Start a gfsh prompt.
2. Connect to a running cluster by either connecting to a locator with an embedded JMX Manager or connect directly to a JMX Manager. For example:

   ```
   gfsh>connect --locator=locator1[10334]
   ```

   or

```
gfsh>connect --jmx-manager=locator1[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 Tanzu GemFire cluster:

GemFire JMX MBean Notifications

This topic explains links to documentation about Tanzu GemFire JMX MBean notifications.

Tanzu GemFire MBeans emit notifications when specific events occur or if an alert is raised in the Tanzu GemFire 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.

- List of JMX MBean Notifications

  This topic lists all available JMX notifications emitted by Tanzu GemFire MBeans.

Notification Federation

This explains notification federation in VMware Tanzu GemFire.

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 cluster. 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 cluster.

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 cluster 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 Tanzu 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 cluster. 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 JMX MBean Notifications

This topic lists all available JMX notifications emitted by VMware Tanzu GemFire MBeans.

Notifications are emitted by the following MBeans:

- MemberMXBean Notifications

- MemberMXBean Gateway Notifications
- CacheServerMXBean Notifications
- DistributedSystemMXBean Notifications

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.async.event.queue.created	Member name or ID	Async Event Queue is Created 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

MemberMXBean Gateway Notifications

Notification Type	Notification Source	Message
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.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

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

This topic explains how to configure VMware Tanzu GemFire RMI registry ports and RMI connectors.

Tanzu GemFire programmatically emulates out-of-the-box JMX provided by Java 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 Tanzu GemFire JMX Manager. To do this, specify values for the jmx-manager-bind-address, which specifies the JMX manager’s IP address and jmx-manager-port, which defines the RMI connection port.

The default Tanzu 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 you need to use standard JMX RMI in your deployment for other monitoring purposes, set the Tanzu 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 Tanzu 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 Tanzu 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

This topic explains how to execute gfsh commands programmatically through the VMware Tanzu GemFire management API.

> 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 class path of your application. Do not copy this library to your class path 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.

> The CommandService API is currently only available on JMX Manager nodes.

```
// Get existing CommandService instance or create new if it does not exist
commandService = CommandService.createLocalCommandService(cache);

// OR simply get CommandService instance if it exists, do not 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");
}
```

Managing Heap and Off-heap Memory

By default, VMware Tanzu GemFire uses the JVM heap. Tanzu GemFire also offers an option to store data off heap. This section describes how to manage heap and off-heap memory to best support your application.

- Heap memory management

  Tips to optimize your application's performance by tuning the way Tanzu GemFire uses the JVM heap.

- Off-heap memory management

  How to configure Tanzu GemFire to store region values in off-heap memory, which is memory within the JVM that is not subject to Java garbage collection.

- Locking memory

  How to prevent the operating system from paging out heap or off-heap memory.

Managing Heap Memory

This topic explains how to manage heap memory when using VMware Tanzu GemFire.

By default, Tanzu GemFire uses the JVM heap. Heap memory is a part of memory allocated to the JVM in which all class instances are allocated. Space allocated to the heap is reclaimed through *garbage collection* (GC), an automatic memory management process.

Using the GemFire Resource Manager

The Tanzu GemFire resource manager works with your JVM's tenured garbage collector to control heap use and protect your member from hangs and crashes due to memory overload.

The Tanzu GemFire resource manager prevents the cache from consuming too much memory by evicting old cached 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.

The resource manager has two threshold settings, each expressed as a percentage of the total tenured heap. Both are deactivated 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.

   The resource manager enforces eviction thresholds only on regions whose LRU eviction policies are based on heap percentage. Regions whose eviction policies are based on entry count or memory size use other mechanisms to manage evictions. See Eviction for more detail regarding eviction policies.

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.

The Critical threshold is enforced on all regions, regardless of LRU eviction policy, though it can be set to zero to deactivate its effect.

When heap use passes the eviction threshold in either direction, the manager logs an info-level message.

When heap use exceeds the critical threshold, the manager logs an error-level message. Avoid exceeding the critical threshold. Once identified as critical, the Tanzu GemFire member becomes a read-only member that refuses cache updates for all of its regions, including incoming distributed updates.

For more information, see org.apache.geode.cache.control.ResourceManager in the online 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.

Heap Size

The most important factor affecting garbage collection performance is total available memory.

The specific heap size you should set depends greatly on the nature of your live data and the rate and nature of the operations on your cache.

In general, you should allocate sufficient space to store all the data in the cache, with additional space recommended to handle bursts and the failures of other nodes. If you lack capacity to store all the data that in the cache, you may run into issues such as GC thrashing and excessive eviction.

Concurrent Mark-Sweep Garbage Collector (CMS) Heaps

To maintain low latency for CMS heaps, we recommend that you initially set the heap size to at least 2 times, and up to 3 times, the size of your live data set.

Z Garbage Collector (ZGC) Heaps

To achieve the 1ms maximum pause time promise of ZGC, you should initially set the heap size to at least 2 times, and up to 3 times, the size of your live data set.

Two key factors affect required heap size with ZGC:

- Java Object Storage: With small heap sizes (less than 32GB), the cache's heap usage can be up to 80% higher with ZGC compared to CMS or G1. The actual impact varies depending on the nature of the Java objects.
- ZGC Headroom: To maintain low latency, ZGC requires additional "headroom," that is, unused heap beyond what is required to store the application's Java objects. The amount of headroom required depends largely on the memory allocation rate, which in turn depends largely on the operations being performed on the cache. Heap sizes should be at least 2 to 3 times the memory usage of the cache itself, which consists of your data plus the GemFire data structures. Memory-intensive operations such as queries can increase this further. The actual headroom required is highly dependent on the rate and nature of the operations on the cache.

Other ZGC Guidance

- Size your data, and determine exactly how much long-lived data you will have in your environment, per member.
- Consider the case when nodes are removed. Determine if you need the system to tolerate one member down, two members down, etc. Decide if you use recovery-delay to restore full redundancy on the remaining n-1, or n-2 members. If you do, expect the maximum heap consumption on the remaining members to be higher than normal until redundancy is recovered.
- Each member will need a heap size that is at least 2 times the maximum expected long-lived heap consumption, potentially up to 3 times if you have very high allocation rates in worst case bursts.
- Use the `Xmx` JVM option to set each member's heap to the required size.
- Set `-XX:SoftMaxHeapSize` slightly below the maximum expected heap consumption to protect the system from any allocation stalls, etc. If you will use GemFire eviction, see Tuning the JVM's Garbage Collection Parameters below for additional guidance for tuning `-XX:SoftMaxHeapSize`.

Controlling Heap Use with the Resource Manager

Resource manager behavior is closely tied to the triggering of Garbage Collection (GC) activities, the use of concurrent garbage collectors 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.

The resource manager is available for use in any Tanzu GemFire member, but you may not want to activate 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 the resource manager consumes. 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.

For the members where you want to activate the resource manager:

1. Configure Tanzu GemFire for heap LRU management.
2. Set the JVM GC tuning parameters to handle heap and garbage collection in conjunction with the Tanzu 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 Tanzu GemFire for Heap LRU Management

The configuration terms used here are cache.xml elements and attributes, but you can also configure through gfsh and the org.apache.geode.cache.control.ResourceManager and Region APIs.

1. When starting up your server, set initial-heap and max-heap to the same value.
2. 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.

   > 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.

3. 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.
4. Decide which regions will participate in heap eviction and set their eviction-attributes to lru-heap-percentage. See Eviction. 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.

gfsh example:

```
gfsh>start server --name=server1 --initial-heap=30m --max-heap=30m \
--critical-heap-percentage=80 --eviction-heap-percentage=60
```

cache.xml example:

```
<cache>
<region refid="REPLICATE_HEAP_LRU" />
...
<resource-manager critical-heap-percentage="80" eviction-heap-percentage="60"/>
</cache>
```

> The resource-manager specification must appear after the region declarations in your cache.xml file.

Tuning the JVM's Garbage Collection Parameters

Because VMware Tanzu GemFire is specifically designed to manipulate data held in memory, you can optimize your application's performance by tuning the way VMware Tanzu GemFire uses the JVM heap.

See your JVM documentation for all JVM-specific settings that can be used to improve garbage collection (GC) response. Best configuration can vary depending on the use case and JVM garbage collector used.

Concurrent Mark-Sweep (CMS) Garbage Collector

If you are using concurrent mark-sweep (CMS) garbage collection with VMware Tanzu GemFire, use the following settings to improve performance:

1. Set the initial and maximum heap parameters, -Xms and -Xmx, to the same values. The gfsh start server options --initial-heap and --max-heap accomplish the same purpose, with the added value of providing resource manager defaults such as eviction threshold and critical threshold.
2. If your JVM allows, configure it to initiate CMS 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 Tanzu 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	CMS switch flag	CMS initiation (begin at heap % N)
Sun HotSpot	-XX:+UseConcMarkSweepGC	-XX:CMSInitiatingOccupancyFraction=N
IBM	-Xgcpolicy:gencon	N/A

For the gfsh start server command, pass these settings with the --J switch, for example: --J=-XX:+UseConcMarkSweepGC.

The following is an example of setting Hotspot JVM for an application:

```
$ java app.MyApplication -Xms=30m -Xmx=30m -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=60
```

> Do not use the -XX:+UseCompressedStrings and -XX:+UseStringCache JVM configuration properties when starting up servers. These JVM options can cause issues with data corruption and compatibility.

Or, using gfsh:

```
$ gfsh start server --name=app.MyApplication --initial-heap=30m --max-heap=30m \
--J=-XX:+UseConcMarkSweepGC --J=-XX:CMSInitiatingOccupancyFraction=60
```

Garbage First (G1) Garbage Collector

Although the Garbage First (G1) garbage collector generally works effectively with GemFire, using G1 with GemFire's heap LRU eviction feature entails particular tradeoffs. While most applications requiring heap LRU eviction are better served with the CMS collector (on older JDKs), or with the ZGC collector (on newer ones), some applications may run best with G1. For instance, for applications needing to run on heaps smaller than 32GB, the G1 garbage collector can prove more memory efficient than ZGC.

The decision to use heap LRU eviction with the G1 garbage collector should be considered carefully. For more information, see LRU Eviction Policy.

Z Garbage Collector (ZGC)

If you are using the Z garbage collector (ZGC), use the following settings to improve performance:

1. Set the initial and maximum heap switches, -Xms and -Xmx, to the same values. Alternatively, use the gfsh start server options --initial-heap and --max-heap to set these values.
2. If you use GemFire heap LRU eviction, configure ZGC to initiate collection when heap use is at least 5% lower than your setting for the resource manager eviction-heap-percentage. The collector should work 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 -XX:SoftMaxHeapSize to a value no higher than 60% of maximum heap size.
3. If you are running generational ZGC (available as of JDK 21, and enabled by default by gfsh) with heap LRU eviction you may need to specify a maximum time between major collection cycles. When GemFire application activity is low, generational ZGC may not perform major collections for long periods of time. This can leave lots of garbage uncollected and can lead to too much eviction activity. If you notice this sort of spurious eviction, experiment with -XX:ZCollectionIntervalMajor=nn to force major GC cycles at a fixed interval (specified in seconds). Shorter periods lead to more garbage collector activity, which in turn can impact GemFire performance. Longer periods reduce the performance impact but can allow more garbage to build up during application lulls, potentially resulting in more eviction than you might like.

With a heap size less than 32 GB, the cache's heap usage may be up to 80% larger with ZGC than with CMS or G1. The magnitude of this impact is highly dependent on the nature of your cached data. For heap sizes of 32 GB or larger, ZGC does not cause this additional overhead.

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 above the critical threshold negatively impacts the entire distributed system. To stay below critical, tune so that the Tanzu 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 Tanzu GemFire ResourceManagerStats provide information about memory use and the manager thresholds and eviction activities.

If your application spikes above the critical threshold on a regular basis, try lowering the eviction threshold. If the application never goes 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 (vendor and version) 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.

Example Scenario

This is one possible scenario for the configuration used in the examples:

- A 64-bit Java VM with 8 Gb of heap space on a 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 OutOfMemoryException errors. Testing has shown that leaving 15% head room above the critical threshold when adding data to the region gives 99.5% uptime with no OutOfMemoryException errors, when configured with the CMS garbage collector using -XX:CMSInitiatingOccupancyFraction=70.

In this next chart, it looks like the manager's evictions are kicking in at the right time, but the CMS garbage collector 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.

- gfsh Example:

```
gfsh>start server --name=server1 --initial-heap=30m --max-heap=30m \
--critical-heap-percentage=85 --eviction-heap-percentage=75
```

```
gfsh>create region --name=bigDataStore --type=PARTITION_HEAP_LRU
```

- XML:

```
<cache>
<region name="bigDataStore" refid="PARTITION_HEAP_LRU"/>
...
<resource-manager critical-heap-percentage="85" eviction-heap-percentage="75"/>
</cache>
```

> The resource-manager specification must appear after the region declarations in your cache.xml file.

- 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");
```

Heap LRU Eviction With G1GC

This topic explains how to configure and tune the Java G1 Garbage Collector (G1GC) to work with the GemFire heap LRU eviction policy on JDK 17 and higher. It outlines key considerations for managing heap usage, avoiding humongous object penalties, and ensuring timely garbage collection to prevent premature or unnecessary data eviction.

There are three eviction policies in GemFire: Heap LRU, entry count, and absolute memory usage.

The LRU Eviction Policy differs from the others because it creates a feedback loop between the GemFire resource manager and the Java garbage collector. As a result, the Java garbage collector must be configured to ensure compatibility. This configuration varies depending on the garbage collector used:

- Concurrent Mark-Sweep (CMS)
- Z Garbage Collector (ZGC)
- G1 Garbage Collector (G1GC).

Configuring G1GC to work with heap LRU eviction can be challenging, but it offers key benefits. One key advantage is that G1GC can be more space-efficient than ZGC for smaller heap sizes.

If you are running GemFire on JDK 17 or higher with a heap size smaller than 32GB, the default configuration might not be ideal. On JDK 17, the gfsh start command selects ZGC by default.

However, for heap sizes under 32 GB, 64-bit JVMs using the CMS or G1GC garbage collectors can benefit from Java's -XX:+UseCompressedOops option, which significantly reduces heap usage. This option is enabled automatically when applicable, but it is not supported with ZGC or with heap sizes of 32 GB or larger.

Depending on your application's requirements, switching to G1GC might improve storage efficiency. However, if you choose to use the G1GC and use the GemFire heap LRU eviction feature, be aware that additional tradeoffs, testing, and tuning will be required.

This topic explains how to configure GemFire and the JDK to support heaps smaller than 32GB on JDK 17 and higher.

Configuring G1GC for GemFire on JDK17

If you need more efficient storage in a small heap on JDK 17, you can disable the ZGC using:

```
-XX:-UseZGC
```

This activates the default G1GC in JDK 17. To explicitly ensure that G1GC is used, you can specify both options:

```
-XX:-UseZGC -XX:+UseG1GC
```

Using G1GC requires additional configuration focused on:

- Managing humongous objects
- Tuning Frequent Garbage Collection to Support Heap LRU Eviction

Managing Humongous Objects

The G1GC divides the heap into regions of a fixed size. Objects larger than half the region size are

considered ‘humongous’ and are stored in special regions called humongous regions. Humongous regions incur space and CPU penalties, so they should be avoided when possible.

> Both GemFire and Java’s G1GC use the term ‘region’. This topic discusses both, so be mindful of which type is being referred to.
>
> - In GemFire, a region is a distributed data structure, essentially a hash map.
> - In G1GC, a region is a memory segment within the Java heap.

Most of the Java heap storage used by a GemFire application is consumed by region map entries. If a region entry exceeds half the G1GC region size, it will be treated as a humongous object and might incur performance penalties associated with humongous objects.

If you are using the G1GC, it is important to determine the maximum size of your region entries. Once you know the size, you can configure G1GC to treat your entries as regular (non-humongous) objects.

For more information on how to calculate the size of your entries, see Memory Requirements for Cached Data.

Setting G1 Region Size
You can ensure entries remain below the humongous threshold by tuning the G1GC region size. To explicitly set the region size:

```
-XX:G1HeapRegionSize=n
```

Where n must be a power of two and can range from 1MB to 32MB. The goal is to have around 2048 regions based on the minimum Java heap size.

If no -XX:G1HeapRegionSize=n option is specified, the JVM automatically chooses an ‘ergonomic’ value. The ergonomic value is approximately the minimum Java heap size divided by 2048 and rounded up to the nearest power of two.

The minimum heap size is specified by the --initial-heap option in the gfsh start command or by the -Xms Java command-line option.

When to Override the Ergonomic Value
If the ergonomic value of G1HeapRegionSize is less than twice the size of your largest GemFire region entry, you should set a larger G1HeapRegionSize. If you are unsure of the value, use

```
-XX:+PrintFlagsFinal
```

Diagnosing Humongous Allocations
To determine the number of regions occupied by humongous objects on the Java heap, enable GC logging:

```
gc+heap=info
```

Look for outputs like:

```
Humongous regions: X->Y
```

Where Y is the number of regions occupied by humongous objects. If Y is greater than zero, some entries exceed the humongous threshold. For more information, see the HotSpot Virtual Machine Garbage Collection Tuning Guide in the Java Platform, Standard Edition Guide.

Tuning Frequent Garbage Collection to Support Heap LRU Eviction

When using the GemFire heap LRU eviction feature with the G1GC, you must tune the collector for compatibility. This tuning will impact application performance and CPU utilization.

GemFire can evict entries from regions when they become too full, helping to prevent heap exhaustion. While this improves resilience, it also increases latency when accessing evicted entries later.

GemFire offers several eviction policies. If your application requires eviction, you should weigh the advantages and disadvantages of each of these three policy choices:

Eviction Policy	Good When
Entry Count	Entry sizes are consistent and known at configuration time
Absolute Memory Usage	Entry sizes may vary. Overhead of ObjectSizer is acceptable
Heap LRU Eviction	Neither entry count nor absolute memory usage policies are acceptable, and you understand the increased importance of realistic testing

Heap LRU eviction sets up a feedback loop between the GemFire resource manager and the Java garbage collector. It relies on the 'current used heap memory' reported by the JVM to decide whether it is time to start evicting entries from regions configured for that.

Regions with lru-heap-percentage set in their eviction attributes begin evicting entries when Java heap memory utilization exceeds a specified threshold. This threshold is defined using the --eviction-heap-percentage option in the gfsh start server command or the eviction-heap-percentage attribute of the resource-manager element in the cache XML. This behavior is known as 'heap LRU eviction', and the eviction-heap-percentage is often referred to as the 'eviction threshold'.

For a detailed explanation of heap LRU eviction, see Managing Heap Memory. For more information on all the policy alternatives, see How Eviction Works.

The following sections outline how to tune heap LRU eviction. If your application doesn't require heap LRU eviction, you can ignore the rest of this section.

Heap Memory Utilization

At any given time, two types of objects occupy space on the Java heap: reachable and unreachable. Unreachable objects can no longer affect program execution and are considered garbage. Reachable objects must be retained.

The garbage collector identifies and recycles garbage based on tradeoffs between CPU, heap space, and latency. Over time, the amount of garbage in the heap fluctuates. By default, the G1GC allows significant variation in the amount of garbage it retains, which can interfere with the GemFire heap LRU eviction algorithm.

How Garbage Affects LRU Eviction

There are two scenarios where excessive garbage accumulation disrupts LRU eviction:

1. During a sudden spike of application activity, if the garbage collector doesn't respond quickly, garbage can accumulate rapidly. This can trigger unnecessary eviction of region entries. A faster garbage collector response could prevent these evictions and subsequent access latency could be avoided.
2. After a busy period, if application activity drops and the garbage collector goes idle, garbage may remain on the heap. This can cause GemFire to continue evicting region entries unnecessarily.

In summary:

Scenario	Garbage Collector might…	Problematic Heap LRU Eviction Result
Sudden spike in application activity	Fail to work fast enough	Premature eviction
Application inactivity after sustained activity	Sit idle	Continued eviction

Goal: Bounded Garbage on Heap

The goal is to maintain a bounded amount of garbage on the heap, regardless of application activity. This becomes increasingly important as heap utilization increases.

- ZGC and CMS provide tuning options that support this goal. For both, the tuning settings can be configured relative to the GemFire eviction-heap-percentage setting.
- For the G1GC, the -XX:G1PeriodicGCInterval setting can support this goal. However, this setting is sensitive to application workload, and determining an appropriate value requires testing.

In summary, these are the primary garbage collector settings for heap LRU eviction:

GC Algorithm	GC Initiation Option (typical)
Concurrent Mark Sweep (CMS)	-XX:CMSInitiatingOccupancyFraction=(eviction-heap-percentage - 10%)
Z Garbage Collector (ZGC)	-XX:SoftMaxHeapSize=(eviction-heap-percentage - 5%)
G1	-XX:G1PeriodicGCInterval=(empirically-derived time period)

-XX:G1PeriodicGCInterval Setting

If there is no garbage collection for a long time because of application inactivity, the VM may hold on to a large amount of unused memory for a long time that could be used elsewhere. To avoid this, G1GC can be forced to do regular garbage collection using the -XX:G1PeriodicGCInterval option. This option determines a minimum interval in ms at which G1GC considers performing a garbage collection.

This setting helps:

- Prevent the garbage collector from sitting idle during periods of application inactivity.
- Avoid premature eviction during sudden spikes in application activity.

For more information, see Periodic Garbage Collections section in the Oracle HotSpot Virtual Machine Garbage Collection Tuning Guide.

Typical -XX:G1PeriodicGCInterval Values

Here are typical -XX:G1PeriodicGCInterval values that have proven useful in addressing the heap LRU eviction pitfalls described above. The optimal interval values for these scenarios vary by application. You will have to choose a setting that offers a workable compromise for your application.

Problem Scenario	Value of -XX:G1PeriodicGCInterval=m to address the problem in isolation (typical)
Premature eviction during sudden spike in application activity	As low as 500 milliseconds
Continual eviction during application inactivity	1000 to 25000 milliseconds

Impact of More Frequent Garbage Collection

Forcing the garbage collector to run more frequently can degrade application performance and increase CPU usage.

Here is the typical relative impact on application performance and CPU utilization

Benchmark	GC Interval (ms)	Throughput Decrease	Latency Increase	CPU Increase
Client-server PUT	500	10%	5%	12%

Benchmark	GC Interval (ms)	Throughput Decrease	Latency Increase	CPU Increase
Peer-to-peer GET	500	10%	14%	8%
Client-server PUT	2000	1%	1%	9%
Peer-to-peer GET	2000	5%	1%	3%
Client-server PUT	10000	1%	1%	13%
Peer-to-peer GET	10000	3%	2%	0%

Where GC Interval = -XX:G1PeriodicGCInterval in milliseconds.

If you decide to use the G1GC on JDK 17, you must test your application and GemFire settings on realistic hardware, with representative application load. This is particularly important if you are using the GemFire heap LRU eviction feature. The following sections show how to use GemFire statistics to troubleshoot common problems.

Troubleshooting Heap LRU Eviction: Waking the Garbage Collector When Your Application is Idle

As a first example, let's examine an application experiencing the 'continual eviction during application inactivity' issue. The following graphs show GemFire statistics from an application using heap LRU eviction. This application performs PUT operations while also running queries. It is tuned so that it PUTs enough entries to exceed the eviction threshold. We expect to see the number of entries in the region to level off at some maximum and remain stable.

First we'll look at a run with no G1GC tuning at all.

When a region is configured with heap LRU eviction, entries should be evicted when the eviction threshold is exceeded. In this case, the application uses the 'overflow to disk' eviction action. As a result, the DiskRegionStatistics tracks both the number of region entries in memory and the number of region entries evicted to disk over time. Here we see that entriesInVM oscillates within a narrow range, while entriesOnlyOnDisk (the evicted entries) is growing.

But when we zoom out to look at the remainder of the run, we see entriesInVM dropping monotonically. This might be reasonable if there were multiple regions and some other regions were consuming heap memory. But we happen to know that is not the case in this application.

What was the application doing at this time? We see it doing PUTs and queries:

But we see that when entriesInVM started declining monotonically, the application had essentially stopped all activity. What is happening here?

Analyzing GC and Heap Behavior

In the next chart we see currentUsedMemory (reported by the G1GC for the “G1 Old Gen” pool) oscillating above and below the GemFire resource manager’s evictionThreshold. But after the oscillation, we see currentUsedMemory stay permanently above the eviction threshold (currentUsedMemory never drops below evictionThreshold in the remainder of this run.)

Why is heap usage not declining? Is there really 7.6GB of heap being consumed by live objects? Or is the garbage collector failing to recycle garbage objects?

Superimposing G1GC collections (for both young and old pools) on our entries chart we see there was no old generation collection activity at all until very late in the run (the single small spike at the right around 16:24:04). And we see that young generation collections were frequent during high application activity, but stopped when application activity stopped.

Fixing the Problem with -XX:G1PeriodicGCInterval

We suspect that the heap does not actually contain 7.6GB of live objects. It is likely that some garbage remains uncollected because the garbage collector is not running frequently enough once application activity decreases. If we can prompt the collector to perform more work during these quieter periods, we may be able to reclaim enough heap space to prevent eviction. To encourage more proactive collection, try setting:

```
-XX:G1PeriodicGCInterval=25000
```

This option causes G1GC to run periodic garbage collections every 25 seconds, even when allocation rates are low. With this setting in place, the decline in entriesInVM was eliminated.

In the chart above, G1 old-Generation currentUsedMemory drops below the evictionThreshold and eviction stops so that entriesInVM are maintained.

The application is actively doing PUTs and queries until about 16:36:31 in this run:

And the cause of the improvement, more frequent GC activity in the young-generation pool after application activity stops, is evident. Notice the two collections at the right of this chart soon after application activity stops. One at 16:37:16 and another at 16:37:41 exactly 25 seconds later.

Troubleshooting Heap LRU Eviction: Ensuring The Garbage Collector Responds Quickly to Sudden Application Activity

We've seen how to address continual eviction during application inactivity. The other common problem scenario is premature eviction during a sudden spike in application activity. When this occurs, you'll typically see entriesInVM dip when application load increases abruptly. In extreme cases, GemFire clients may encounter a LowMemoryException, which indicates that the critical-heap-percentage has been exceeded. Setting a lower value for -XX:G1PeriodicGCInterval can often resolve the issue.

Response to Sudden Application Activity

Let's examine the behavior of an application that populates a region with entries via PUT. During this phase, we expect no eviction. After that, the application runs queries that generate garbage.

Here we see currentUsedMemory in the 'G1 Old Gen' pools of one cache server during a period when the application is issuing all its PUT operations. The eviction threshold has been exceeded, triggering eviction. This application stops issuing PUTs when it notices eviction has started.

The total memory available to the Java VM is 1048MB. Eviction heap percentage is 40% or 419MB. Region entries in this application are all about 1MB in size. If there was no garbage on the Java heap, we would expect to be able to store close to 419 region entries in-memory on a cache server before exceeding the eviction threshold. While the exact number may vary, we should be able to get reasonably close.

Examine how many entries were stored on this server that exceeded its eviction threshold. The total number of entries in memory on the cache server with PID 2330 settles out to 40 at 16:18:36. That's far short of our ideal capacity of 419.

Let's see how many entries were evicted (to disk):

A total of 105 entries were evicted. The sum of resident plus evicted entries is 145. If the garbage collector was doing a better job, we would have been able to retain them all in memory.

The next chart shows a couple of 15ms/s spikes of young-generation GC activity. This level of garbage collection is not sufficient to eliminate garbage quickly enough to prevent premature eviction.

Setting -XX:G1PeriodicGCInterval

Next we'll examine the effect of setting -XX:G1PeriodicGCInterval=750 (three-quarters of a second) on this application.

With this setting, the application can issue all its PUTs without currentUsedMemory exceeding evictionThreshold. After the final PUT, currentUsedMemory gradually increases as the application starts to issue queries, which produce garbage.

During this time period, how many entries did we store in memory? We have 154 in memory after all PUTs are complete versus only 40 in the previous run:

And we see no (0) entries on disk in this time period:

currentUsedMemory was lower in this case because Young-generation GC activity was much more intense than in the previous run. Instead of a couple of short 15ms/s spikes, we now see up to 83ms/s of collectionTime, starting well before the PUTs begin. There is also measurable GC activity in every single sample period:

Managing Off-Heap Memory

This topic explains how to manage off-heap memory in VMware Tanzu GemFire.

You can configure Tanzu GemFire to store region values in *off-heap memory*, memory within the JVM that is not subject to Java garbage collection.

Garbage collection (GC) within a JVM can prove to be a performance impediment. A server cannot exert control over when garbage collection within the JVM heap memory takes place, and the server has little control over the triggers for invocation. Off-heap memory offloads values to a storage area that is not

subject to Java GC. By taking advantage of off-heap storage, an application can reduce the amount of heap storage that is subject to GC overhead.

Off-heap memory works in conjunction with the heap, it does not replace it. The keys are stored in heap memory space. Tanzu GemFire's own memory manager handles the off-heap memory with better performance than the Java garbage collector would for certain sets of region data.

The resource manager monitors the contents of off-heap memory and invokes memory management operations in accordance with two thresholds similar to those used for monitoring the JVM heap: eviction-off-heap-percentage and critical-off-heap-percentage.

On-heap and Off-heap Objects

The following objects are always stored in the JVM heap:

- Region metadata
- Entry metadata
- Keys
- Indexes
- Subscription queue elements

The following objects can be stored in off-heap memory:

- Values - maximum value size is 2GB
- Reference counts
- List of free memory blocks
- WAN queue elements

> Do not use functional range indexes with off-heap data, as they are not supported. An attempt to do so generates an exception.

Off-heap Recommendations

Off-heap storage is best suited to data patterns where:

- Stored values are relatively uniform in size
- Stored values are mostly less than 128K in size
- The usage patterns involve cycles of many creates followed by destroys or clear
- The values do not need to be frequently deserialized
- Many of the values are long-lived reference data

Be aware that Tanzu GemFire has to perform extra work to access the data stored in off-heap memory since it is stored in serialized form. This extra work may cause some use cases to run slower in an off-heap configuration, even though they use less memory and avoid garbage collection overhead. However, even with the extra deserialization, off-heap storage may give you the best performance. Features that may increase overhead include

- frequent updates
- stored values of widely varying sizes

- deltas
- queries

Implementation Details

The off-heap memory manager is efficient at handling region data values that are all the same size or are of fixed sizes. With fixed and same-sized data values allocated within the off-heap memory, freed chunks can often be re-used, and there is little or no need to devote cycles to defragmentation.

Region values that are less than or equal to eight bytes in size will not reside in off-heap memory, even if the region is configured to use off-heap memory. These very small size region values reside in the JVM heap in place of a reference to an off-heap location. This performance enhancement saves space and load time.

Controlling Off-heap Use with the Resource Manager

The Tanzu GemFire resource manager controls off-heap memory by means of two thresholds, in much the same way as it does JVM heap memory. See Using the Tanzu GemFire Resource Manager. The resource manager prevents the cache from consuming too much off-heap memory by evicting old data. If the off-heap memory manager is unable to keep up, the resource manager refuses additions to the cache until the off-heap memory manager has freed an adequate amount of memory.

The resource manager has two threshold settings, each expressed as a percentage of the total off-heap memory. Both are deactivated by default.

1. Eviction Threshold. The percentage of off-heap memory at which eviction should begin. Evictions continue until the resource manager determines that off-heap memory use is again below the eviction threshold. Set the eviction threshold with the eviction-off-heap-percentage region attribute. The resource manager enforces an eviction threshold only on regions with the HEAP_LRU characteristic. If critical threshold is non-zero, the default eviction threshold is 5% below the critical threshold. If critical threshold is zero, the default eviction threshold is 80% of total off-heap memory.

   The resource manager enforces eviction thresholds only on regions whose LRU eviction policies are based on heap percentage. Regions whose eviction policies based on entry count or memory size use other mechanisms to manage evictions. See Eviction for more detail regarding eviction policies.

2. Critical Threshold. The percentage of off-heap memory at which the cache is at risk of becoming inoperable. When cache use exceeds the critical threshold, all activity that might add data to the cache is refused. Any operation that would increase consumption of off-heap memory throws a LowMemoryException instead of completing its operation. Set the critical threshold with the critical-off-heap-percentage region attribute.

   Critical threshold is enforced on all regions, regardless of LRU eviction policy, though it can be set to zero to deactivate its effect.

Specifying Off-heap Memory

To use off-heap memory, specify the following options when setting up servers and regions:

- Start the JVM as described in Tuning the JVM's Garbage Collection Parameters. In particular, set the initial and maximum heap sizes to the same value. Sizes less than 32GB are optimal when you plan to use off-heap memory.

- From gfsh, start each server that will support off-heap memory with a non-zero off-heap-memory-size value, specified in megabytes (m) or gigabytes (g). If you plan to use the resource manager, specify critical threshold, eviction threshold, or (in most cases) both.

  Example:

  ```
  gfsh> start server --name=server1 —initial-heap=10G —max-heap=10G -—off-heap-memory-size=200G \
  —lock-memory=true —critical-off-heap-percentage=90 —eviction-off-heap-percentage=80
  ```

- Mark regions whose entry values should be stored off-heap by setting the off-heap region attribute to true Configure other region attributes uniformly for all members that host data for the same region. .

  Example:

  ```
  gfsh>create region --name=region1 --type=PARTITION_HEAP_LRU --off-heap=true
  ```

gfsh Off-heap Support

gfsh supports off-heap memory in server and region creation operations and in reporting functions:

- alter disk-store: --off-heap=(true | false) resets the off-heap attribute for the specified region. See alter disk-store for details.
- create region: --off-heap=(true | false) sets the off-heap attribute for the specified region. See create region for details.
- describe member: Displays off-heap size.
- describe offline-disk-store: Shows if an off-line region is off-heap.
- describe region: Displays the value of a region's off-heap attribute.
- show metrics: Includes off-heap metrics maxMemory, freeMemory, usedMemory, objects, fragmentation and defragmentationTime.
- start server: Supports off-heap options --lock-memory, --off-heap-memory-size, --critical-off-heap-percentage, and --eviction-off-heap-percentage See start server for details.

ResourceManager API

The org.apache.geode.cache.control.ResourceManager interface defines methods that support off-heap use:

- public void setCriticalOffHeapPercentage(float Percentage)
- public float getCriticalOffHeapPercentage()
- public void setEvictionOffHeapPercentage(float Percentage)
- public float getEvictionOffHeapPercentage()

The gemfire.properties file supports one off-heap property:

off-heap-memory-size: Specifies the size of off-heap memory in megabytes (m) or gigabytes (g). For example:

```
off-heap-memory-size=4096m
off-heap-memory-size=120g
```

See gemfire.properties and gfsecurity.properties (Tanzu GemFire Properties) for details.

The cache.xml file supports one region attribute:

off-heap(=true | false): Specifies that the region uses off-heap memory; defaults to false. For example:

```
<region-attributes
  off-heap="true">
</region-attributes>
```

See <region-attributes> for details.

The cache.xml file supports two resource manager attributes:

critical-off-heap-percentage=value: Specifies the percentage of off-heap memory at or above which the cache is considered in danger of becoming inoperable due to out of memory exceptions. See <resource-manager> for details.

eviction-off-heap-percentage=value: Specifies the percentage of off-heap memory at or above which eviction should begin. Can be set for any region, but actively operates only in regions configured for HEAP_LRU eviction. See <resource-manager> for details.

For example:

```
<cache>
...
  <resource-manager
    critical-off-heap-percentage="99.9"
    eviction-off-heap=-percentage="85"/>
  ...
</cache>
```

Tuning Off-heap Memory Usage

Tanzu GemFire collects statistics on off-heap memory usage which you can view with the gfsh show metrics command. See Off-Heap (OffHeapMemoryStats) for a description of available off-heap statistics.

Off-heap memory is optimized, by default, for storing values of 128 KB in size. This figure is known as the "maximum optimized stored value size," which we will denote here by *maxOptStoredValSize*. If your data typically runs larger, you can enhance performance by increasing the OFF_HEAP_FREE_LIST_COUNT system parameter to a number larger than maxOptStoredValSize/8, where *maxOptStoredValSize* is expressed in KB (1024 bytes). So, the default values correspond to:

```
128 KB / 8 = (128 * 1024) / 8 = 131,072 / 8 = 16,384
-Dgemfire.OFF_HEAP_FREE_LIST_COUNT=16384
```

To optimize for a maximum optimized stored value size that is twice the default, or 256 KB, the free list count should be doubled:

```
-Dgemfire.OFF_HEAP_FREE_LIST_COUNT=32768
```

During the tuning process, you can toggle the off-heap region attribute on and off, leaving other off-heap settings and parameters in place, in order to compare your application's on-heap and off-heap performance.

Locking Memory (Linux Systems Only)

This topic explains how to lock memory on a Linux system when using VMware Tanzu GemFire.

On Linux systems, you can lock memory to prevent the operating system from paging out heap or off-heap memory.

To use this feature:

1. Configure the operating system limits for locked memory. Increase the operating system's ulimit -l value (the maximum size that may be locked in memory) from the default (typically 32 KB or 64 KB) to at least the total amount of memory used by Tanzu GemFire for on-heap or off-heap storage. To view the current setting, enter ulimit -a at a shell prompt and find the value for max locked memory:

   ```
   # ulimit -a
   ...
   max locked memory       (kbytes, -l) 64
   ...
   ```

   Use ulimit -l max-size-in-kbytes to raise the limit. For example, to set the locked memory limit to 64 GB:

   ```
   # ulimit -l 64000000
   ```

2. Using locked memory in this manner increases the time required to start Tanzu GemFire. The additional time required to start Tanzu GemFire depends on the total amount of memory used, and can range from several seconds to 10 minutes or more. To improve startup time and reduce the potential of member timeouts, instruct the kernel to free operating system page caches just before starting a Tanzu GemFire member by issuing the following command:

   ```
   $ echo 1 > /proc/sys/vm/drop_caches
   ```

3. Start each Tanzu GemFire data store with the gfsh -lock-memory=true option. If you deploy more than one server per host, begin by starting each server sequentially. Starting servers sequentially avoids a race condition in the operating system that can cause failures (even machine crashes) if you accidentally over-allocate the available RAM. After you verify that the system configuration is stable, you can then start servers concurrently.

Disk Storage

This topic links to documentation about disk storage in VMware Tanzu GemFire.

With Tanzu 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

  Overflow and persistence use disk stores individually or together to store data.

- 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.

- 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, VMware Tanzu GemFire shrinks the files to the space used in each file.

- Configuring Disk Stores

In addition to the disk stores you specify, VMware Tanzu 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.

- Optimizing a System with Disk Stores

  Optimize availability and performance by following the guidelines in this section.

- Start Up and Shut Down with Disk Stores

  This section describes what happens during startup and shutdown and provides procedures for those operations.

- 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.

- Creating Backups for System Recovery and Operational Management

  A backup is a copy of persisted data from a disk store. A backup is used to restore the disk store to the state it was in when the backup was made. The appropriate back up and restore procedures differ based upon whether the cluster is online or offline. An online system has currently running members. An offline system does not have any running members.

How Disk Stores Work

This topic explains how VMware Tanzu GemFire disk stores work.

Overflow and persistence use disk stores individually or together to store data.

Disk storage is available for these items:

- Regions. Persist and/or overflow data from 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 Tanzu GemFire PDX serialization.

Each member has its own set of disk stores, and they are completely separate from the disk stores of any other member. For each disk store, 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 Tanzu GemFire Writes to the Disk Store

Tanzu GemFire writes the following to the disk store:

- Persisted and overflowed data as specified when the disk store was created and configured
- The members that host the store and information about their status, such as which members are online and which members are offline and time stamps
- A disk store identifier
- Which regions are in the disk store, specified by region name and including selected attributes
- Names of colocated regions on which the regions in the disk store depend
- A record of all operations on the regions

Tanzu GemFire does not write indexes to disk.

Disk Store State

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

Disk store access and management differs according to whether the member is online or offline. While a member is running, its disk stores are online. When the member exits and is not running, its disk stores are offline.

- 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.
- Offline, the disk store is just a collection of files in the host file system. The files are accessible based on file system permissions. You can copy the files for backup or to move the member's disk store location. You can also run some maintenance operations, such as file compaction and validation, by using the gfsh command-line interface. When offline, the disk store's information is unavailable to the cluster. For partitioned regions, region data is split between multiple members, and therefore the start up of a member is dependent on all members, and must wait for all members to be online. An attempt to access an entry that is stored on disk by an offline member results in a PartitionOfflineException.

Disk Store File Names and Extensions

This topic describes VMware Tanzu GemFire 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: usage identifier, disk store name, and oplog sequence number.

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
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 -tlr persistData1/
total 8
-rw-rw-r--   1 person users        188 Mar  4 06:17 BACKUPpersistDS1.if
-rw-rw-r--   1 person users          0 Mar  4 06:18 BACKUPpersistDS1_1.drf
-rw-rw-r--   1 person users         38 Mar  4 06:18 BACKUPpersistDS1_1.crf

bash-2.05$ ls -tlr overflowData1/
total 1028
-rw-rw-r--   1 person users          0 Mar  4 06:21 DRLK_IFoverflowDS1.lk
-rw-rw-r--   1 person users          0 Mar  4 06:21 BACKUPoverflowDS1.if
-rw-rw-r--   1 person users 1073741824 Mar  4 06:21 OVERFLOWoverflowDS1_1.crf
```

Example default disk store files for a persistent region:

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

Disk Store Operation Logs

This topic explains VMware Tanzu GemFire 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, Tanzu GemFire shrinks the files to the space used in each file.

After the oplog is closed, Tanzu 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 Tanzu GemFire to load the entry values in the background after the entry keys are loaded.

When an operation log is full, Tanzu 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.

> 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. Tanzu 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 person users       512 Mar 22 13:56 ..
-rw-rw-r--   1 person users         0 Mar 22 13:57 BACKUPCacheOverflow_2.drf
-rw-rw-r--   1 person users    426549 Mar 22 13:57 BACKUPCacheOverflow_2.crf
-rw-rw-r--   1 person users         0 Mar 22 13:57 BACKUPCacheOverflow_1.drf
-rw-rw-r--   1 person users    936558 Mar 22 13:57 BACKUPCacheOverflow_1.crf
-rw-rw-r--   1 person users      1924 Mar 22 13:57 BACKUPCacheOverflow.if
drwxrwxr-x   2 person 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 Oplogs Reach the Configured Disk Capacity

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

- If auto-compaction is enabled, Tanzu 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.
  ```

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

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

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

Configuring Disk Stores

This topic provides links to documentation that explains how to configure VMware Tanzu GemFire disk stores.

In addition to the disk stores that you specify, Tanzu 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.

- Defining 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.

- Disk Store Configuration Parameters

  You define your disk stores by using the gfsh create disk-store command or 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.

- 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”.

Defining and Configuring Disk Stores

This topic explains how to define and configure disk stores in VMware Tanzu GemFire.

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.

> Besides the disk stores that you specify, Tanzu 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. For information about changing the behavior, see Create and Configure Your Disk Stores and Modifying the Default Disk Store.

- Design Your Disk Stores
- Create and Configure Your Disk Stores
- Configuring Regions, Queues, and PDX Serialization to Use the Disk Stores
- Configuring Disk Stores on Gateway Senders

Design Your Disk Stores

Before you begin, review GemFire Basic Configuration and Programming.

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.
    - When calculating your disk requirements, figure in your data modification patterns and your compaction strategy. Tanzu GemFire creates each oplog file at the max-oplog-size, which defaults to 100 MiB. Obsolete operations are removed from the oplogs only 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 / (compaction_threshold / 100) ) * data_size) + (max_oplog_size * segments)
      ```

      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 temporarily above the upper bound. If you deactivate 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. Tanzu 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. Open a gfsh prompt and connect to the cluster.
3. At the gfsh prompt, create and configure a disk store:
    - Specify the name (--name) of the disk-store.
        - 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.

    ```
    gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data
    ```

    - Configure the directory locations (--dir) and the maximum space to use for the store (specified after the disk directory name by # and the maximum number in megabytes).

    ```
    gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data
    ```

    - Optionally, you can configure the store's segments. Example:

    ```
    gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data --segments=10
    ```

    - Optionally, you can configure the store's file compaction behavior. In conjunction with this, plan and program for any manual compaction. Example:

    ```
    gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data --segments=10 \
    --compaction-threshold=40 --auto-compact=false --allow-force-compaction=true
    ```

    - If needed, configure the maximum size (in MB) 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:

    ```
    gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data --segments=10 \
    --compaction-threshold=40 --auto-compact=false --allow-force-compaction=true \
    --max-oplog-size=512
    ```

- If 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 (number of operations) or time-interval (milliseconds) is reached, enqueued data is flushed to disk. You can also synchronously flush unwritten data to disk through the DiskStore flushToDisk method. Example:

```
gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data --segments=10 \
--compaction-threshold=40 --auto-compact=false --allow-force-compaction=true \
--max-oplog-size=512 --queue-size=10000 --time-interval=15
```

- If needed, modify the size (specified in bytes) of the buffer used for writing to disk. Example:

```
gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data --segments=10 \
--compaction-threshold=40 --auto-compact=false --allow-force-compaction=true \
--max-oplog-size=512 --queue-size=10000 --time-interval=15 --write-buffer-size=65536
```

- If needed, modify the disk-usage-warning-percentage and disk-usage-critical-percentage thresholds that determine the percentage (default: 90%) of disk usage that will trigger a warning and the percentage (default: 99%) of disk usage that will generate an error and shut down the member cache. Example:

```
gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data --segments=10 \
--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-usage-warning-percentage=80 --disk-usage-critical-percentage=98
```

The following is the complete disk store cache.xml configuration example:

```
<disk-store name="serverOverflow" segments="10" 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-usage-warning-percentage="80"
   disk-usage-critical-percentage="98">
   <disk-dirs>
      <disk-dir>c:\overflow_data</disk-dir>
      <disk-dir dir-size="20480">d:\overflow_data</disk-dir>
   </disk-dirs>
</disk-store>
```

> As an alternative to defining cache.xml on every server in the cluster: If the cluster configuration service is enabled, when you create a disk store in gfsh, you can share the disk store's configuration with the rest of cluster. See Overview of the Cluster Configuration Service.

Modifying Disk Stores

You can modify an offline disk store by using the alter disk-store command. If you are modifying the default disk store configuration, use "DEFAULT" as the disk-store name.

Configuring Regions, Queues, and PDX Serialization to Use the Disk Stores

The following are examples of using already created and named disk stores for Regions, Queues, and PDX Serialization.

Example of using a disk store for region persistence and overflow:

- gfsh:

```
gfsh>create region --name=regionName --type=PARTITION_PERSISTENT_OVERFLOW \
--disk-store=serverPersistOverflow
```

- cache.xml

```
<region refid="PARTITION_PERSISTENT_OVERFLOW" disk-store-name="persistOverflow1"/>
```

Example of using a named disk store for server subscription queue overflow (cache.xml):

```
<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 (cache.xml):

```
<pdx read-serialized="true"
   persistent="true"
   disk-store-name="SerializationDiskStore">
</pdx>
```

Configuring Disk Stores on Gateway Senders

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.

Example of using a named disk store for a serial gateway sender queue persistence:

- gfsh:

```
gfsh>create gateway-sender --id=persistedSender1 --remote-distributed-system-id=1 \
--enable-persistence=true --disk-store-name=diskStoreA --maximum-queue-memory=100
```

- cache.xml:

```
<cache>
  <gateway-sender id="persistedsender1" parallel="true"
   remote-distributed-system-id="1"
   enable-persistence="true"
   disk-store-name="diskStoreA"
   maximum-queue-memory="100"/>
   ...
</cache>
```

Examples of using the default disk store for a serial gateway sender queue persistence and overflow:

- gfsh:

```
gfsh>create gateway-sender --id=persistedSender1 --remote-distributed-system-id=1 \
--enable-persistence=true --maximum-queue-memory=100
```

- cache.xml:

```
<cache>
  <gateway-sender id="persistedsender1" parallel="true"
   remote-distributed-system-id="1"
   enable-persistence="true"
   maximum-queue-memory="100"/>
   ...
</cache>
```

Segmented Disk Stores

Introduced in GemFire version 10.1, the segmented disk stores feature optimizes read, write and recovery performance by spreading regions and partitioned region buckets across segments. You can think of segments as subsidiary disk stores. Spreading the regions across segments can reduce the contention for a single shared disk store. The number of segments may be configured when a disk store is created or upgraded. If unspecified, the number of segments is determined by the number of CPUs available. Benchmarking has shown that this number provides the best performance with the least number of open files in most cases. In some cases it may be necessary to use a different number of segments. If disk store performance is not optimal with the default value, profile your application with different segment counts.

The additional segments change the disk allocation pattern from what was expected from the non-segmented disk stores. Since each segment has its own set of oplogs there is the potential for more disk usage compared to that of an unsegmented disk store. The compaction-threshold is considered on each set of segment oplogs independently.

Disabling Segmented Disk Stores

VMware recommends that you do not disable segmented disk stores unless absolutely required.

To disable segmented disk stores, and fall back to the orignal non-segmented disk store, set the Java system property gemfire.disk.disableSegmentedDiskStore=true on any server using disk stores.

```
gfsh> start server ... --J=-Dgemfire.disk.disableSegmentedDiskStore=true
```

Alternatively, you may set the environment variable GEMFIRE_DISABLE_SEGMENTED_DISK_STORE=true before starting any processes. Each subprocess should inherit this environment variable.

```
GEMFIRE_DISABLE_SEGMENTED_DISK_STORE=true gfsh
gfsh> start server ...
```

Disk Store Configuration Parameters

This topic lists and explains the disk store configuration parameters in VMware Tanzu GemFire.

You define your disk stores by using the gfsh create disk-store command or 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 gfsh create disk-store command-line parameters as well as getter and setter methods in the org.apache.geode.cache.DiskStoreFactory and org.apache.geode.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.	DEFAULT
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 its live data content percentage drops below the compaction-threshold.	true
compaction-threshold	Percentage (0..100) of live data (non-garbage content) remaining in the operation log, below which it is eligible for compaction. As garbage is created (by entry destroys, entry updates, and region destroys and creates) the percentage of remaining live data declines. Falling below this percentage initiates compaction if auto-compaction is turned on. If not, the file will be eligible for manual compaction at a later time.	50
disk-usage-critical-percentage	Disk usage above this threshold generates an error message and shuts down the member's cache. For example, if the threshold is set to 99%, then falling under 10 GB of free disk space on a 1 TB drive generates the error and shuts down the cache. Set to "0" (zero) to deactivate.	99
disk-usage-warning-percentage	Disk usage above this threshold generates a warning message. For example, if the threshold is set to 90%, then on a 1 TB drive falling under 100 GB of free disk space generates the warning. Set to "0" (zero) to deactivate.	90
max-oplog-size	Maximum size, in megabytes, for an oplog file. When the oplog file reaches this size, the file is rolled over to a new file.	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 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.	0

disk-store attribute	Description	Default
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, in bytes, used to write to disk.	32768
segments	Number of segments to organize regions and buckets into.	CPU count

disk-store sub-element	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 sub-elements to the disk-dirs element. The data is spread evenly among the active disk files in the directories, keeping within any limits you set.

Example:

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

> The directories must exist when the disk store is created or the system throws an exception. Tanzu 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

This topic explains how to modify the default VMware Tanzu GemFire 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 stores without specifying the disk store to use, Tanzu 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.

Examples of using the default disk store for region persistence and overflow:

- gfsh:

```
gfsh>create region --name=regionName --type=PARTITION_PERSISTENT_OVERFLOW
```

- cache.xml

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

Example of using the default disk store for server subscription queue overflow (cache.xml):

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

Change the Behavior of the Default Disk Store

Tanzu 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 that cannot be changed about the default disk store is the name.

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

cache.xml:

```
<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>
```

Optimizing a System with Disk Stores

This topic explains how to optimize the availability and performance of a VMware Tanzu GemFire system with disk store.

1. Tanzu GemFire recommends the use of ext4 filesystems when operating on Linux or Solaris platforms. The ext4 filesystem supports preallocation, which benefits disk startup performance. If you are using ext3 filesystems in latency-sensitive environments with high write throughput, you can improve disk startup performance by setting the maxOplogSize (see the DiskStoreFactory.setMaxOplogSize) to a value lower than the default 1 GB and by deactivating preallocation by specifying the system property gemfire.preAllocateDisk=false upon Tanzu GemFire process startup.
2. 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.
3. Shut down your system using the gfsh shutdown command. This is an ordered shutdown that positions your disk stores for a faster startup.

4. Configure critical usage thresholds (disk-usage-warning-percentage and disk-usage-critical-percentage) for the disk. By default, these are set to 80% for warning and 99% for errors that will shut down the cache.
5. Decide on a file compaction policy and, if needed, develop procedures to monitor your files and execute regular compaction.
6. Decide on a backup strategy for your disk stores and follow it. You can back up a running sytem by using the backup disk-store command.
7. If you remove any persistent region or change its configuration while your disk store is offline, consider synchronizing the regions in your disk stores.
8. By default, the disk stores are recovered in parallel when the cluster restarts. Parallel disk store recovery is usually faster than sequential disk store recovery. If you require sequential disk store recovery, start your servers by setting the system property: geode.parallelDiskStoreRecovery=false. For example, using gfsh to start your servers:

```
start server --name=server1 --J=-Dgeode.parallelDiskStoreRecovery=false
```

Start Up and Shut Down with Disk Stores

This topic describes what happens during VMware Tanzu GemFire startup and shutdown and provides procedures for those operations.

Start Up

When you start a member with a persistent region, the data is retrieved from disk stores to recreate the member's persistent region. If the member does not hold all of the most recent data for the region, then other members have the data, and region creation blocks, waiting for the those other members. A partitioned region with colocated entries also blocks on start up, waiting for the entries of the colocated region to be available. A persistent gateway sender is treated the same as a colocated region, so it can also block region creation.

With a log level of info or below, the system provides messaging about the wait. Here, the disk store for server2 has the most recent data for the region, and server1 is waiting for server2.

```
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: /192.0.2.0:/home/dsmith/server1/.

Members with potentially new data:
[
  DiskStore ID: 160d415538c44ab0-9f7d97bae0a2f8de
  Name: server2
  Location: /192.0.2.0:/home/dsmith/server2/.
]
Use the `gfsh show missing-disk-stores` command to see all disk stores
that are being waited on by other members.
```

When the most recent data is available, the system updates the region, logs a message, and continues the 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 member's disk store has data for a region that is never created, the data remains in the disk store.

Each member's persistent regions load and go online as quickly as possible, not waiting unnecessarily for other members to complete. For performance reasons, these actions occur asynchronously:

- Once at least one copy of each and every bucket is recovered from disk, the region is available. Secondary buckets will load asynchronously.
- Entry keys are loaded from the key file in the disk store before considering entry values. Once all keys are loaded, Tanzu GemFire loads the entry values asynchronously. If a value is requested before it has loaded, the value will immediately be fetched from the disk store.

Start Up Procedure

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. Make sure to start members that host colocated regions, as well as persistent gateway senders.

   While they are initializing their regions, the members determine which have the most recent region data, and initialize their regions with the most recent data.

   For replicated regions, where you define persistence only in some of the region's host members, start the persistent replicate members prior to the non-persistent replicate members 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. It exits with an error status if one of the jobs fails.

   ```
   #!/bin/bash
   ssh servera "cd /my/directory; gfsh start server --name=servera &
   ssh serverb "cd /my/directory; gfsh start server --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 blocked members. When a member blocks waiting for more recent data from another member, the member waits indefinitely rather than coming online with stale data. Check for missing disk stores with the `gfsh show missing-disk-stores` command. See Handling Missing Disk Stores.

   - If no disk stores are missing, the cache initialization must be slow for some other reason. See the information about member hangs in Diagnosing System Problems.
   - If disk stores are missing that you think should be there:
     - Make sure you have started the member. Check the logs for any failure messages. See Logging.

        - 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.
    - 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.

Example Startup to Illustrate Ordering

The following lists the two possibilities for starting up a replicated persistent region after a shutdown. Assume that Member A (MA) exits first, leaving persisted data on disk for RegionP. Member B (MB) continues to run operations on RegionP, which update its disk store and leave the disk store for MA in a stale condition. MB exits, leaving the most up-to-date data on disk for RegionP.

- Restart order 1
    1. MB is started first. MB identifies that it has the most recent disk data for RegionP and initializes the region from disk. MB does not block.
    2. MA is started, recovers its data from disk, and updates region data as needed from the data in MB.
- Restart order 2
    1. MA is started first. MA identifies that it does not have the most recent disk data and blocks, waiting for MB to start before recreating RegionP in MA.
    2. MB is started. MB identifies that it has the most recent disk data for RegionP and initializes the region from disk.
    3. MA recovers its RegionP data from disk and updates region data as needed from the data in MB.

Shutdown

If more than one member hosts a persistent region or queue, the order in which the various members shut down may be significant upon restart of the system. The last member to exit the system or shut down has the most up-to-date data on disk. Each member knows which other system members were online at the time of exit or shutdown. This permits a member to acquire the most recent data upon subsequent start up.

For a replicated region with persistence, the last member to exit has the most recent data.

For a partitioned region every member persists its own buckets. A shutdown using gfsh shutdown will synchronize the disk stores before exiting, so all disk stores hold the most recent data. Without an orderly shutdown, some disk stores may have more recent bucket data than others.

The best way to shut down a system is to invoke the gfsh shutdown command with all members running. All online data stores will be synchronized before shutting down, so all hold the most recent data copy. To shut down all members other than locators:

```
gfsh>shutdown
```

To shut down all members, including locators:

```
gfsh>shutdown --include-locators=true
```

Disk Store Management

This topic links to documentation that explains how to manage VMware Tanzu GemFire disk stores.

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 for a list of available commands.

- Disk Store Management Commands and Operations
- Validating a Disk Store
- Running Compaction on Disk Store Log Files
- Keeping a Disk Store Synchronized with the Cache
- Configuring Disk Free Space Monitoring
- Handling Missing Disk Stores
- Altering When Buffers Are Flushed to Disk

  You can configure Tanzu GemFire to write immediately to disk and you may be able to modify your operating system behavior to perform buffer flushes more frequently.

Disk Store Management Commands and Operations

This topic explains how to manage your disk stores using the gfsh command-line tool. For more information about gfsh commands, see gfsh and Disk Store Commands.

> 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	Keeping a Disk Store Synchronized with the Cache
compact disk-store	On	Running Compaction on Disk Store Log Files
backup disk-store	On	Creating Backups for System Recovery and Operational Management
compact offline-disk-store	Off	Running Compaction on Disk Store Log Files
export offline-disk-store	Off	Creating Backups for System Recovery and Operational Management
revoke missing-disk-store	On	Handling Missing Disk Stores
show missing-disk-stores	On	Handling Missing Disk Stores
shutdown	On	Start Up and Shut Down with Disk Stores
validate offline disk-store	Off	Validating a Disk Store

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 cluster 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

This topic explains how to validate a VMware Tanzu GemFire 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 is 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

This topic explains how to run compaction on VMware Tanzu GemFire 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 Tanzu 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. Tanzu GemFire does not remove garbage records as it goes, but it tracks the percentage of non-garbage (live data) remaining in each operation log, and provides mechanisms for removing garbage to compact your log files.

Tanzu 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.

The system is configured by default to automatically compact any closed operation log when its non-garbage content drops below a certain percentage. This automatic compaction is well suited to most Tanzu GemFire implementations. In some circumstances, you may choose to manually initiate compaction for online and offline disk stores.

Log File Compaction for the Online Disk Store

For the online disk store, the current operation log is not available for compaction, no matter how much garbage it contains. You can use DiskStore.forceRoll to close the current oplog, making it eligible for compaction. See Disk Store Operation Logs for details.

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

Run Online Compaction

Old log files become eligible for online compaction when their live data (non-garbage) content drops below 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, Tanzu GemFire automatically compacts each oplog when its non-garbage (live data) content drops below the compaction-threshold. This takes cycles from your other operations, so you may want to deactivate 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 Tanzu GemFire to maintain extra data about the files so it can compact on demand. This is deactivated 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. Tanzu 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.findDiskStore("myDiskStore").forceCompaction();
```

- Using gfsh, compact a disk store with the compact disk-store command. Examples:

```
gfsh>compact disk-store --name=Disk1

gfsh>compact disk-store --name=Disk1 --group=MemberGroup1,MemberGroup2
```

> 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 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
```

> 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, Tanzu 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 deactivate 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-compacted disk data. You might need to increase system monitoring to make sure you do not overrun your disk space. You may be able to run only offline compaction. If so, you can set allow-force-compaction to false and avoid storing the information required for manual online compaction.

Directory Size Limits

Reaching directory size limits during compaction 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 user users         3 Apr  7 14:56 BACKUPds1_3.drf
-rw-rw-r--   1 user users        25 Apr  7 14:56 BACKUPds1_3.crf
drwxrwxr-x   3 user users      1024 Apr  7 15:02 ..
-rw-rw-r--   1 user users      7085 Apr  7 15:06 BACKUPds1.if
-rw-rw-r--   1 user users        18 Apr  7 15:07 BACKUPds1_4.drf
-rw-rw-r--   1 user users      1070 Apr  7 15:07 BACKUPds1_4.crf
drwxrwxr-x   2 user 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 user users         3 Apr  7 14:56 BACKUPds1_3.drf
-rw-rw-r--   1 user users        25 Apr  7 14:56 BACKUPds1_3.crf
drwxrwxr-x   3 user users      1024 Apr  7 15:02 ..
-rw-rw-r--   1 user users         0 Apr  7 15:08 BACKUPds1_5.drf
-rw-rw-r--   1 user users       638 Apr  7 15:08 BACKUPds1_5.crf
-rw-rw-r--   1 user users      2788 Apr  7 15:08 BACKUPds1.if
drwxrwxr-x   2 user users       512 Apr  7 15:09 .
bash-2.05$
```

Keeping a Disk Store Synchronized with the Cache

This topic explains how to keep a disk store synchronized with the cache in VMware Tanzu GemFire.

Recovering data from an offline disk store proceeds most quickly when the configuration of the offline data matches that of the online data.

Whenever you change or remove persistent regions (by modifying your cache.xml or the code that configures the regions), then you should alter the corresponding offline disk-store to match. If you do not, then the next time this disk-store is recovered it will recover all of that region's data into a temporary region using the old configuration. The old configuration will still consume the old configured resources (heap memory, off-heap memory). If those resources are no longer available (for example the old configuration of the region was off-heap but you decide to no longer configure off-heap memory on the JVM), the disk-store recovery will fail.

It is common practice to have more than one off-line disk store, because each member of the cluster usually has its own copy. Be sure to apply the same alter disk-store command to each offline copy of the

disk store.

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 a subset of the region configuration attributes. (For a list of the retained attributes, see alter disk-store). 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.

For example, to change the initial capacity of the region named “partitioned_region” in the disk store:

```
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 the command with no actions specified:

```
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

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.

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

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

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

You can remove the region from the disk store 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 start it. Exercise caution when removing the files from the file system, as more than one region can be specified to use the same disk store directories.
- Selectively remove the discontinued region from the disk store with a command such as:

  ```
  gfsh>alter disk-store --name=myDiskStoreName --region=partitioned_region
  --disk-dirs=/firstDiskStoreDir,/secondDiskStoreDir,/thirdDiskStoreDir --remove
  ```

To guard against unintended data loss, Tanzu 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.

Configuring Disk Free Space Monitoring

This table explains how to configure disk free space monitoring in VMware Tanzu GemFire.

To modify disk-usage-warning-percentage and disk-usage-critical-percentage thresholds, specify the parameters when executing the gfsh create disk-store command.

```
gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data#20480 \
--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-usage-warning-percentage=80 --disk-usage-critical-percentage=98
```

By default, disk usage above 80% triggers a warning message. Disk usage above 99% generates an error and shuts down the member cache that accesses that disk store. To deactivate disk store monitoring, set the parameters to 0.

To view the current threshold values set for an existing disk store, use the gfsh describe disk-store command:

```
gfsh>describe disk-store --member=server1 --name=DiskStore1
```

You can also use the following DiskStoreMXBean method APIs to configure and obtain these thresholds programmatically.

- getDiskUsageCriticalPercentage
- getDiskUsageWarningPercentage
- setDiskUsageCriticalPercentage
- setDiskUsageWarningPercentage

You can obtain statistics on disk space usage and the performance of disk space monitoring by accessing the following statistics:

- diskSpace
- maximumSpace
- volumeSize
- volumeFreeSpace
- volumeFreeSpaceChecks
- volumeFreeSpaceTime

See Disk Space Usage (DiskDirStatistics).

Handling Missing Disk Stores

This topic explains how to how to handle missing disk stores in VMware Tanzu GemFire. This topic 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 PartitionOfflineException in cache operations or preventing the system from satisfying redundancy.

Example:

```
gfsh>show missing-disk-stores
        Disk Store ID             |  Host    |              Directory
------------------------------------ | --------- | ------------------------------------
60399215-532b-406f-b81f-9b5bd8d1b55a | excalibur | /usr/local/gemfire/deploy/disk_store1
```

> You need to be connected to JMX Manager in gfsh to run this command.
>
> 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 cluster 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.

> 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
```

Altering When Buffers Are Flushed to Disk

This topic explains how to alter when buffers are flushed to disk in VMware Tanzu GemFire.

You can configure Tanzu GemFire to write immediately to disk and you may be able to modify your operating system behavior to perform buffer flushes more frequently.

Typically, Tanzu 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 Tanzu 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.

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 about 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 Tanzu GemFire to Flush Buffers on Disk Writes

You can have Tanzu 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 Tanzu GemFire member. You can only modify this setting when you start a member. When this is set, Tanzu 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 Tanzu 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:

```
gfsh>start server --name=... --J=-Dgemfire.syncWrites=true
```

Creating Backups for System Recovery and Operational Management

This topic explains how to create backups for VMware Tanzu GemFire system recovery and operational management.

A backup is a copy of persisted data from a disk store. A backup is used to restore the disk store to the state it was in when the backup was made. The appropriate back up and restore procedures differ based upon whether the cluster is online or offline. An online system has currently running members. An offline system does not have any running members.

- Making a Backup While the System Is Online
- What a Full Online Backup Saves
- What an Incremental Online Backup Saves
- Disk Store Backup Directory Structure and Contents
- Offline Members—Manual Catch-Up to an Online Backup
- Restore Using a Backup Made While the System Was Online

Making a Backup While the System is Online

The gfsh command backup disk-store creates a backup of the disk stores for all members running in the cluster. The backup works by passing commands to the running system members. This requires the members to be online for this operation to succeed. Each member with persistent data creates a backup of its own configuration and disk stores. The backup does not block any activities within the cluster, but it does use resources.

> When connected to a GemFire version 10.1 or later locator, if members running a GemFire version earlier than 10.1 exist in the cluster, issuing the backup disk-store command results in the following error message: "Backup only supports GemFire 10.1 and later. The following members are running on an earlier GemFire version."
> When connected to a GemFire version 10.0 or earlier locator, the backup disk-store runs successfully even if other locators have been upgraded to 10.1 or later.

> Do not try to create backup files from a running system by using your operating system's file copy commands. This creates incomplete and unusable copies.

Preparing to Make a Backup

- Consider compacting your disk store before making a backup. If auto-compaction is turned off, you may want to do a manual compaction to save on the quantity of data copied over the network by the backup. For more information about configuring a manual compaction, see Manual Compaction.
- Take the backup when region operations are quiescent, to avoid the possibility of an inconsistency between region data and an asynchronous event queue (AEQ) or a WAN Gateway sender (which uses a persistent queue). A region operation that causes a persisted write to a region involves a disk operation. The associated queue operation also causes a disk operation. These two disk operations are not made atomically, so if a backup is made between the two disk operations, then the backup represents inconsistent data in the region and the queue.
- 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 cluster, and it can affect performance.
- Configure each member with any additional files or directories to be backed up by modifying the member's cache.xml file. Additional items that ought to be included in the backup:
  - application jar files
  - other files that the application needs when starting, such as a file that sets the classpath

  For example, to include file myExtraBackupStuff in the backup, the cache.xml file specification of the data store would include:

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

  Directories are recursively copied, with any disk stores that are found excluded from this user-specified backup.
- 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 all members to write to the directory and create subdirectories.

The directory specified for the backup can be used multiple times. Each time a backup is made, a new subdirectory is created within the specified directory, and that new subdirectory's name represents the date and time.

You can use one of two locations for the backup:

- a single physical location, such as a network file server, for example:

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

- a directory that is local to all host machines in the system, for example:

```
./gemfireBackupLocation
```

- Make sure all members with persistent data are running in the system, because offline members cannot back up their disk stores. Output from the backup command will not identify members hosting replicated regions that are offline.

How to Do a Full Online Backup

1. If auto-compaction is deactivated, and manual compaction is needed:

```
gfsh>compact disk-store --name=Disk1
```

2. Run the gfsh backup disk-store command, specifying the backup directory location. For example:

```
gfsh>backup disk-store --dir=/export/fileServerDirectory/gemfireBackupLocation
```

The output will list information for each member that has successfully backed up disk stores. The tabular information will contain the member's name, its UUID, the directory backed up, and the host name of the member.

Any online member that fails to complete its backup will leave a file named INCOMPLETE_BACKUP in its highest level backup directory. The existence of this file identifies that the backup file contains only a partial backup, and it cannot be used in a restore operation.

3. Validate the backup for later recovery use. On the command line, each backup can be checked with commands such as

```
cd 2010-04-10-11-35/straw_14871_53406_34322/diskstores/ds1
gfsh validate offline-disk-store --name=ds1 --disk-dirs=/home/dsmith/dir1
```

How to Do an Incremental Backup

An incremental backup contains items that have changed since a previous backup was made.

To do an incremental backup, specify the backup directory that the incremental backup will be based upon with the --baseline-dir argument. For example:

```
gfsh>backup disk-store --dir=/export/fileServerDirectory/gemfireBackupLocation
--baseline-dir=/export/fileServerDirectory/gemfireBackupLocation/2012-10-01-12-30
```

The output will appear the same as the output for a full online backup.

Any online member that fails to complete its incremental backup will leave a file named INCOMPLETE_BACKUP in its highest level backup directory. The existence of this file identifies that the backup file contains only a partial backup, and it cannot be used in a restore operation. The next time a backup is made, a full backup will be made.

What a Full Online Backup Saves

For each member with persistent data, a full backup includes the following:

- Disk store files for all members containing persistent region data.
- Files and directories specified in the cache.xml configuration file as <backup> elements. For example:

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

- Deployed JAR files that were deployed using the gfsh deploy 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 primary 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, called restore.bat on Windows, and called restore.sh on Linux. This script may later be used to do a restore. The script copies files back to their original locations.

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

```
$ cd thebackupdir
$ ls -R
./2023-10-18-13-44-53:
server1/ server2/

./2023-10-18-13-44-53/server1:
config/ diskstores/ README.txt restore.sh user/

./2023-10-18-13-44-53/server1/config:
cache.xml

./2023-10-18-13-44-53/server1/diskstores:
DEFAULT/

./2023-10-18-13-44-53/server1/diskstores/DEFAULT:
dir0/

./2023-10-18-13-44-53/server1/diskstores/DEFAULT/dir0:
BACKUPDEFAULT.if 0/ 1/

./2023-10-18-13-44-53/server1/diskstores/DEFAULT/dir0/0:
BACKUPDEFAULT_0_1.crf BACKUPDEFAULT_0_1.drf

./2023-10-18-13-44-53/server1/diskstores/DEFAULT/dir0/1:
BACKUPDEFAULT_1_1.crf BACKUPDEFAULT_1_1.drf

./2023-10-18-13-44-53/server1/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. Bring this member's files into the online backup framework manually, and create a restore script by hand starting with 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 other than the restore script.
4. Copy in this member's files.
5. Modify the restore script to work for this member.

Restore Using a Backup Made While the System Was Online

The restore.sh or restore.bat script copies files back to their original locations.

1. Restore your disk stores while cache members are offline and the system is down.
2. Look at each of the restore scripts to see where they will place the files and make sure the destination locations are ready. A restore script will refuse to copy over files with the same names.
3. Run each restore 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.

Cache and Region Snapshots

This topic explains cache and region snapshots in VMware Tanzu GemFire and links to documentation about 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 into another, such as capturing data from a production system and moving it into a smaller QA or development system.

In effect, you can load data from one cluster into another cluster. Administrators export a snapshot of a region or an entire cache (multiple regions) and later import the snapshot into another region or cluster 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.

> 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.

- Exporting Cache and Region Snapshots

  To save Tanzu GemFire cache or region data to a snapshot that you can later load into another cluster or region, use the cache.getSnapshotService.save API, region.getSnapshotService.save API, or the gfsh command-line interface (export data).

- Importing Cache and Region Snapshots

  To import a Tanzu GemFire cache or region data snapshot that you previously exported into another cluster or region, use the cache.getSnapshotService.load API, region.getSnapshotService.load API, or the gfsh command-line interface (import data).

- 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.

- Reading Snapshots Programmatically

  You can read a snapshot entry-by-entry for further processing or transformation into other formats.

Usage and Performance Notes

This topic explains how to optimize the VMware Tanzu GemFire 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, we recommend 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

This topic explains how to export VMware Tanzu GemFire cache and regions snapshots.

To save Tanzu GemFire cache or region data to a snapshot that you can later load into another cluster 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 for an example.

Exporting a Region Snapshot

You can also export a specific region by using the API or gfsh commands below.

> In the case of non-persistent regions, the snapshot that you export contains both in-cache entries and entries that overflow to disk.

Java API:

```
File mySnapshot = ...
Region<String, MyObject> region = ...

region.getSnapshotService().save(mySnapshot, SnapshotFormat.GEMFIRE);
```

gfsh:

Open a gfsh prompt. After connecting to a Tanzu GemFire cluster, 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 that hosts the region. 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 about this command, see export data.

Export Example with Options

These examples show how to include the parallel option for exporting partitioned regions. Note that the parallel option takes a directory rather than a file; see export data for details.

Java API:

```
File mySnapshotDir = ...
Region<String, MyObject> region = ...

SnapshotOptions<Integer, MyObject> options =
 region.getSnapshotServive.createOptions().setParallelMode(true);
region.getSnapshotService().save(mySnapshotDir, SnapshotFormat.GEMFIRE, options);
```

gfsh:

The Java API example, above, accomplishes the same purpose as the following gfsh command:

```
gfsh>export data --parallel --region=region1 --dir=region1_2012_10_10 --member=server1
```

Importing Cache and Region Snapshots

This topic explains how to import VMware Tanzu GemFire cache and region snapshots.

To import a Tanzu GemFire cache or region data snapshot that you previously exported into another cluster 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.

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 Tanzu GemFire cluster, 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 a member that hosts the region. 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 about this command, see import data. For an example of how to invoke this command with additional options, see Export Example with Options.

Filtering Entries During Import or Export

This topic explains how to customize your VMware Tanzu GemFire 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 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 = snapsrv.createOptions().setFilter(even);

// only save cache entries with an even key
snapsrv.save(mySnapshot, SnapshotFormat.GEMFIRE, options);
```

Reading Snapshots Programmatically

This topic explains how read a VMware Tanzu GemFire snapshot programatically.

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();
}
```

Region Compression

This section describes region compression in VMware Tanzu GemFire, and the benefits and use of compression.

One way to reduce memory consumption by Tanzu GemFire is to enable compression in your regions. Tanzu GemFire allows you to compress in-memory region values using pluggable compressors (compression codecs). Tanzu GemFire includes the Snappy compressor as the built-in compression codec; however, you can implement and specify a different compressor for each compressed region.

What Gets Compressed

When you enable compression in a region, all values stored in the region are compressed while in memory. Keys and indexes are not compressed. New values are compressed when put into the in-memory cache and all values are decompressed when being read from the cache. Values are not compressed when persisted to disk. Values are decompressed before being sent over the wire to other peer members or clients.

When compression is enabled, each value in the region is compressed, and each region entry is compressed as a single unit. It is not possible to compress individual fields of an entry.

You can have a mix of compressed and non-compressed regions in the same cache.

- Guidelines on Using Compression

  This topic describes factors to consider when deciding on whether to use compression.

- How to Enable Compression in a Region

  This topic describes how to enable compression on your region.

- Working with Compressors

  When using region compression, you can use the default Snappy compressor included with Tanzu GemFire or you can specify your own compressor.

- Comparing Performance of Compressed and Non-Compressed Regions

  The comparative performance of compressed regions versus non-compressed regions can vary depending on how the region is being used and whether the region is hosted in a memory-bound JVM.

Guidelines to Using Compression

This topic describes factors to consider when deciding on whether to use compression.

Review the following guidelines when deciding on whether or not to enable compression in your region:

- Use compression when JVM memory usage is too high. Compression allows you to store more region data in-memory and to reduce the number of expensive garbage collection cycles that prevent JVMs from running out of memory when memory usage is high.

  To determine if JVM memory usage is high, examine the following statistics:

  - vmStats>freeMemory
  - vmStats->maxMemory
  - ConcurrentMarkSweep->collectionTime

  If the amount of free memory regularly drops below 20% to 25% or the duration of the garbage collection cycles is generally on the high side, then the regions hosted on that JVM are good candidates for having compression enabled.

- Consider the types and lengths of the fields in the region's entries. Since compression is performed on each entry separately (and not on the region as a whole), consider the potential for duplicate data across a single entry. Duplicate bytes are compressed more easily. Also, since region entries are first serialized into a byte area before being compressed, how well the data might compress is determined by the number and length of duplicate bytes across the entire entry and not just a single field. Finally, the larger the entry the more likely compression will achieve good results as the potential for duplicate bytes, and a series of duplicate bytes, increases.

- Consider the type of data you wish to compress. The type of data stored has a significant impact on how well the data may compress. String data will generally compress better than numeric data simply because string bytes are far more likely to repeat; however, that may not always be the case. For example, a region entry that holds a couple of short, unique strings may not provide as much memory savings when compressed as another region entry that holds a large number of integer values. In short, when evaluating the potential gains of compressing a region, consider the likelihood of having duplicate bytes, and more importantly the length of a series of duplicate bytes, for a single, serialized region entry. In addition, data that has already been compressed, such as JPEG format files, can actually cause more memory to be used.

- Compress if you are storing large text values. Compression is beneficial if you are storing large text values (such as JSON or XML) or blobs in Tanzu GemFire that would benefit from compression.

- Consider whether fields being queried against are indexed. You can query against compressed regions; however, if the fields you are querying against have not been indexed, then the fields must be decompressed before they can be used for comparison. In short, you may incur some query performance costs when querying against non-indexed fields.

- Objects stored in the compression region must be serializable. Compression only operates on byte arrays, therefore objects being stored in a compressed region must be serializable and deserializable. The objects can either implement the Serializable interface or use one of the other Tanzu GemFire serialization mechanisms (such as PdxSerializable). Implementers should always be aware that when compression is enabled the instance of an object put into a region will not be the same instance when taken out. Therefore, transient attributes will lose their value when the containing object is put into and then taken out of a region.

- Compressed regions will enable cloning by default. Setting a compressor and then deactivating cloning results in an exception. The options are incompatible because the process of compressing/serializing and then decompressing/deserializing will result in a different instance of the object being created and that may be interpreted as cloning the object.

How to Enable Compression in a Region

This topic describes how to enable compression on your region.

To enable compression on your region, set the following region attribute in your cache.xml:

```
<?xml version="1.0" encoding= "UTF-8"?>
<cache xmlns="http://geode.apache.org/schema/cache"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cache-1.0.xsd"
 version="1.0" lock-lease="120"  lock-timeout= "60" search-timeout= "300"  is-server= "true"  copy-on-read= "false" >
 <region name="compressedRegion" >
  <region-attributes data-policy="replicate" ... />
   <compressor>
    <class-name>org.apache.geode.compression.SnappyCompressor</class-name>
   </compressor>
   ...
  </region-attributes>
 </region>
</cache>
```

In the Compressor element, specify the class-name for your compressor implementation. This example specifies the Snappy compressor, which is bundled with Tanzu GemFire . You can also specify a custom compressor. See Working with Compressors for an example.

Compression can be enabled during region creation using gfsh or programmatically as well.

Using gfsh:

```
gfsh>create-region --name="CompressedRegion" --compressor="org.apache.geode.compression.SnappyCompressor";
```

API:

```
regionFactory.setCompressor(new SnappyCompressor());
```

or

```
regionFactory.setCompressor(SnappyCompressor.getDefaultInstance());
```

How to Check Whether Compression is Enabled

You can also check whether a region has compression enabled by querying which codec is being used. A null codec indicates that no compression is enabled for the region.

```
Region myRegion = cache.getRegion("myRegion");
Compressor compressor = myRegion.getAttributes().getCompressor();
```

Working with Compressors

When using region compression, you can use the default Snappy compressor included with Tanzu GemFire or you can specify your own compressor.

The compression API consists of a single interface that compression providers must implement. The default compressor (SnappyCompressor) is the single compression implementation that comes bundled with the product. Note that since the Compressor is stateless, there only needs to be a single instance in any JVM; however, multiple instances may be used without issue. The single, default instance of the SnappyCompressor may be retrieved with the SnappyCompressor.getDefaultInstance() static method.

> The Snappy codec included with Tanzu GemFire cannot be used with Solaris deployments. Snappy is only supported on Linux, Windows, and macOS deployments of Tanzu GemFire.

This example provides a custom Compressor implementation:

```
package com.mybiz.myproduct.compression;

import org.apache.geode.compression.Compressor;

public class LZWCompressor implements Compressor {
  private final LZWCodec lzwCodec = new LZWCodec();

  @Override
  public byte[] compress(byte[] input) {
      return lzwCodec.compress(input);
  }

  @Override
  public byte[] decompress(byte[] input) {
      return lzwCodec.decompress(input);
  }
}
```

To use the new custom compressor on a region:

1. Make sure that the new compressor package is available in the classpath of all JVMs that will host the region.
2. Configure the custom compressor for the region using any of the following mechanisms:

   Using gfsh:

   ```
   gfsh>create-region --name="CompressedRegion" \
   --compressor="com.mybiz.myproduct.compression.LZWCompressor"
   ```

   Using API:

   For example:

   ```
   regionFactory.setCompressor(new LZWCompressor());
   ```

   cache.xml:

   ```
   <region-attributes>
     <Compressor>
       <class-name>com.mybiz.myproduct.compression.LZWCompressor</class-name>
     </Compressor>
   </region-attributes>
   ```

Changing the Compressor for an Already Compressed Region

You typically enable compression on a region at the time of region creation. You cannot modify the Compressor or deactivate compression for the region while the region is online.

However, if you need to change the compressor or deactivate compression, you can do so by performing the following steps:

1. Shut down the members hosting the region you wish to modify.
2. Modify the cache.xml file for the member either specifying a new compressor or removing the compressor attribute from the region.
3. Restart the member.

Comparing Performance of Compressed and Non-Compressed Regions

The comparative performance of compressed regions versus non-compressed regions can vary depending on how the region is being used and whether the region is hosted in a memory-bound JVM.

When considering the cost of enabling compression, you should consider the relative cost of reading and writing compressed data as well as the cost of compression as a percentage of the total time spent managing entries in a region. As a general rule, enabling compression on a region will add 30% to 60% more overhead for region create and update operations than for region get operations. Because of this, enabling compression will create more overhead on regions that are write heavy than on regions that are read heavy.

However, when attempting to evaluate the performance cost of enabling compression you should also consider the cost of compression relative to the overall cost of managing entries in a region. A region may be tuned in such a way that it is highly optimized for read and/or write performance. For example, a replicated region that does not save to disk will have much better read and write performance than a partitioned region that does save to disk. Enabling compression on a region that has been optimized for read and write performance will provide more noticeable results than using compression on regions that have not been optimized this way. More concretely, performance may degrade by several hundred percent on a read/write optimized region whereas it may only degrade by 5 to 10 percent on a non-optimized region.

A final note on performance relates to the cost when enabling compression on regions in a memory bound JVM. Enabling compression generally assumes that the enclosing JVM is memory bound and therefore spends a lot of time for garbage collection. In that case performance may improve by as much as several hundred percent as the JVM will be running far fewer garbage collection cycles and spending less time when running a cycle.

Monitoring Compression Performance

The following statistics provide monitoring for cache compression:

- compressTime
- decompressTime
- compressions
- decompressions
- preCompressedBytes
- postCompressedBytes

See Cache Performance (CachePerfStats) for statistic descriptions.

Network Partitioning

This topic links to documentation VMware Tanzu GemFire architecture and management features. Use these features to detect and resolve network partition problems.

- How Network Partitioning Management Works

  Tanzu GemFire handles network outages by using a weighting system to determine whether the remaining available members have a sufficient quorum to continue as a cluster.

- Failure Detection and Membership Views

  Tanzu GemFire uses failure detection to remove unresponsive members from membership views.

- 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.

- Network Partitioning Scenarios

  This topic describes network partitioning scenarios and what happens to the partitioned sides of the cluster.

- Configure VMware Tanzu GemFire to Handle Network Partitioning

  This section lists the configuration steps for network partition detection.

- Preventing Network Partitions

  This section provides a short list of things you can do to prevent network partition from occurring.

How Network Partitioning Management Works

This topic explains how VMware Tanzu GemFire network partitioning management works.

Tanzu GemFire handles network outages by using a weighting system to determine whether the remaining available members have a sufficient quorum to continue as a cluster.

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 cluster 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. When a partitioned system is detected, only one side of the system keeps running and the other side automatically shuts down.

The network partitioning detection feature is enabled by default with a true value for the enable-network-partition-detection property. See Configure VMware Tanzu GemFire to Handle Network Partitioning for details. Quorum weight calculations are always performed and logged regardless of this configuration setting.

The overall process for detecting a network partition is as follows:

1. The cluster starts up. When you start up a cluster, start the locators first, start the cache servers second, and then start other members such as applications or processes that access cluster data.
2. After the members start up, the oldest member, typically a locator, assumes the role of the membership coordinator. Peer discovery occurs as members come up and members generate a membership discovery list for the cluster. 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 cluster:
   - Member processes make a request to the coordinator to join the cluster. If authenticated, the coordinator creates a new membership view, hands the new membership view to the new member, and begins the process of sending the new membership 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 leaving or being removed through the normal failure detection process. Failure detection removes unresponsive or slow members. See Managing Slow Receivers and Failure Detection and Membership Views for descriptions of the failure detection process. If a new membership view is sent out that includes one or more failed processes, the coordinator will log the new weight calculations. At any point, if quorum loss is detected due to unresponsive processes, the coordinator 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 cluster), the coordinator generates a new membership view. The membership view is generated by a two-phase protocol:

   1. In the first phase, the membership coordinator sends out a view preparation message to all members and waits for a view preparation acknowledgement from each member. If the coordinator does not receive an ack message from a member (within a specified timeout period – see below), the coordinator attempts to connect to the member's failure-detection socket. If the coordinator cannot connect to the member's failure-detection socket, the coordinator declares the member dead and starts the membership view protocol again from the beginning.

      The timeout period for acknowledgement of a view change, the ack view timeout period, is based on the value of the member-timeout system property, and defaults to about 12 seconds (12437ms). The allowable range for the view ack timeout setting is 1500ms to 12437ms.

   2. In the second phase, the coordinator sends out the new membership view to all members that acknowledged the view preparation message or passed the connection test.

5. Each time the membership coordinator sends a view, each member 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.
   - 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 fail over to the new coordinator, then the new coordinator will have to determine these losses during the view preparation step.

6. With a default value of enable-network-partition-detection, any member that detects a loss of quorum declares a network partition event. A loss of quorum happens when half or more of the member weight is lost in a single view change. In that case, the coordinator sends a network-partitioned-detected message to all members (even to the non-responsive ones) and then closes the cluster

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 comprise a quorum will continue operations. The surviving members elect a new coordinator, designate a lead member, and so on.

Failure Detection and Membership Views

This topic explains how VMware Tanzu 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 has each member observe messages from the peer to its right within the membership view (see Membership Views below for the view layout).

A member that suspects the failure of its peer to the right sends a heartbeat request to the suspect member. With no response from the suspect member, the suspicious member broadcasts a SuspectMembersMessage message to all other members.

The coordinator attempts to connect to the suspect member. If the connection attempt is unsuccessful, the suspect member is removed from the membership view.

The suspect member is sent a message to disconnect from the cluster and close the cache. In parallel to the receipt of the SuspectMembersMessage, a distributed algorithm promotes the leftmost member within the view to act as the coordinator, if the coordinator is the suspect member.

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.

> The TCP connection ping is not used for connection keep alive purposes. It is only used to detect failed members. For a TCP keep alive configuration, see TCP/IP KeepAlive Configuration.

If a new membership view is sent out that includes one or more failed members, the coordinator will log new quorum weight calculations. At any point, if quorum loss is detected due to unresponsive processes, the coordinator will also log a severe level message to identify the failed members:

```
Possible loss of quorum detected due to loss of {0} cache processes: {1}
```

in which {0} is the number of processes that failed and {1} lists the members (cache processes).

Membership Views

The following is a sample membership view:

```
[info 2012/01/06 11:44:08.164 PST bridgegemfire1 <Membership Messenger Non-Blocking> tid=0x1f]
Membership: received new view: 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 port used for membership communication. The distribution-port is the TCP/IP port that is used for cache messaging.
- Each member watches the member to its right for failure detection purposes.

Membership Coordinators, Lead Members, and Member Weighting

This topic describes membership coordinators, lead members, and member weighing in VMware Tanzu GemFire.

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 cluster. With network partition detection enabled, the coordinator can be any Tanzu GemFire member but locators are preferred. In a locator-based system, if all locators are in the reconnecting state, the system continues to function, but new members are not able to join until a locator has successfully reconnected. After a locator has reconnected, the reconnected locator will take over the role of coordinator.

When a coordinator is shutting down, it sends out a view that removes itself from the 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 total weight of crashed members. For the coordinator to continue in its role, it must maintain communication with the majority (by weight) of members. If the total weight of crashed members is greater than or equal to half of the original member weight, then the majority is not reachable and a network partition is declared. When a network partition is declared, the membership coordinator initiates a shutdown.

Sample Member Weight Calculations

This section provides some example calculations. Here are some view-change scenarios from the perspective of the coordinator, in each example the coordinator is a locator:

Example 1: Cluster with 12 members. 2 locators, 10 cache servers. One cache server is designated as lead member (as always). One locator is the coordinator. View total weight equals 111. If members weighing 55.5 or more crash then the majority is lost.

- 4 cache servers become unreachable. Total membership weight loss is 40. The cluster stays up.
- 1 locator (not the coordinator) and 4 cache servers (including the lead member) become unreachable. Membership weight loss equals 48. The cluster stays up.
- 5 cache servers (including the lead member) and 1 locator become unreachable. Membership weight loss equals 58. Majority (weight) is lost, so the coordinator initiates shutdown.

Example 2: Cluster with 4 members. 2 locators, 2 cache servers. One cache server is designated lead member. One locator is the coordinator. View total weight is 31. If members weighing 15.5 or more crash then the majority is lost.

- Cache server designated as lead member becomes unreachable. Membership weight loss equals 15. Cluster stays up.
- Cache server designated as lead member and 1 locator become unreachable. Member weight loss equals 18. Majority (weight) is lost, so the membership coordinator 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 VMware Tanzu GemFire network partitioning scenarios and what happens to the partitioned sides of the cluster.

network partitioning scenario described below

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 message to the members. The coordinator then closes its cluster 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’s callback. 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.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 cannot 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 cluster 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 Tanzu GemFire to Handle Network Partitioning

This topic explains how to configure VMware Tanzu GemFire to handle network partition.

The system uses a combination of member coordinators and system members, designated as lead members, to detect and resolve network partitioning problems.

- Network partition detection works in all environments. Using multiple locators mitigates the effect of network partitioning. See Configuring Peer-to-Peer Discovery.

- Network partition detection is enabled by default. The default setting in the gemfire.properties file is

  ```
  enable-network-partition-detection=true
  ```

  Processes that do not have network partition detection enabled are not eligible to be the lead member, so their failure will not trigger declaration of a network partition.

  All system members should have the same setting for enable-network-partition-detection. If they do not, the system throws a GemFireConfigException upon startup.

- The property enable-network-partition-detection must be true if you are using either partitioned or persistent regions. If you create a persistent region and enable-network-partition-detection to set to false, you will receive the following warning message:

  ```
  Creating persistent region {0} but enable-network-partition-detection is set to false.
  Running with network partition detection deactivated can lead to an unrecoverable system in the event of a network spl
  it.
  ```

- Configure regions you want to protect from network partitioning with a scope setting of DISTRIBUTED_ACK or GLOBAL. Do not use DISTRIBUTED_NO_ACK scope. This prevents operations from being performed throughout the cluster before a network partition is detected. Note: Tanzu 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.
  ```

- 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. Note: The weight loss calculation uses round to nearest. Therefore, a value of 50.51 is rounded to 51 and will cause a network partition.
  - Failure detection is initiated if a member's ack-wait-threshold (default is 15 seconds) and ack-severe-alert-threshold (15 seconds) properties elapse 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 pool read-timeout should be set to at least three times the member-timeout setting in the server's gemfire.properties file. The default 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.

- By default, members that are forced out of the cluster by a network partition event will automatically restart and attempt to reconnect. Data members will attempt to reinitialize the cache. See Handling Forced Cache Disconnection Using Auto-reconnect.

Preventing Network Partitions

This topic provides actions to take to prevent network partitions from occurring when using VMware Tanzu GemFire.

- Use NIC teaming for redundant connectivity. For more information, see Using NIC Teaming for Connectivity Redundancy in *VMware Infrastructure 3 in a Cisco Network Environment* in the Cisco documentation on the Cisco website.
- VMware recommends that 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.
- Configure the weighing of Tanzu GemFire members. For example, you could assign important processes a higher weight.

Security

This topic links to documentation security in VMware Tanzu GemFire.

The security framework permits authentication of connecting components and authorization of operations for all communicating components of the cluster.

- Security Implementation Introduction and Overview

  Encryption, SSL secure communication, authentication, and authorization help to secure the cluster.
- Hardening Guide

  This section gathers discrete details in one convenient location to better help you assess and configure the security of your environment.
- Enable Security with Property Definitions
- Authentication

  A cluster using authentication bars malicious peers or clients, and deters inadvertent access to its cache.
- 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.
- Post Processing of Region Data
- SSL/TLS

  SSL/TLS protects your data in transit between applications.

Security Implementation Introduction and Overview

This topic provides an introduction and overview to VMware Tanzu GemFire security implementation.

Security Features

Encryption, SSL secure communication, authentication, and authorization features help to secure the cluster.

Security features include:

- A single security interface for all components. The single authentication and authorization mechanism simplifies the security implementation. It views and interacts with all components in a

consistent manner.

- System-wide role-based access control. Roles regiment authorized operations requested by the various components.
- SSL communication. Allows configuration of connections to be SSL-based, rather than plain socket connections. You can enable SSL separately for peer-to-peer, client, JMX, gateway senders and receivers, and HTTP connections.
- Post processing of region data. Return values for operations that return region values may be formatted.

Overview

An authentication and authorization mechanism forms the core of the internal security of the cluster. Communications may be further protected by enabling SSL for data in transit.

Authentication verifies the identity of communicating components, leading to control over participation. The variety of participants include peer members, servers, clients, originators of JMX operations, Pulse, gateway senders and receivers representing WAN members of the system, and commands arriving from gfsh on behalf of system users or administrators.

Connection requests trigger the invocation of an authentication callback. This special-purpose callback is written as part of the application, and it attempts to authenticate the requester by whatever algorithm it chooses. The result is either a returned principal representing the requester's authenticated identity or an exception indicating that the requester has not been authenticated. The principal becomes part of any request for operations, which go through the authorization process.

Given authentication, isolation and access to cache data and system state can be further protected by implementing the authorization mechanism, also implemented as a special-purpose callback as part of the application. For example, the protection may be to permit only certain system administrators to start and stop servers. The authority to do this needs to be limited to specific verified accounts, preventing those without the authorization. An implementation of the authorization callback will require that an authenticate identity accompanies all requests to the system, and that the system maintains a representation of which identities are permitted to complete which actions or cache commands.

Hardening Guide

This section describes steps you can take to help you assess and configure the security of your environment in order to protect your VMware Tanzu GemFire cluster and your application from security intrusions.

Hardening Recommendations

The following security measures are strongly recommended for most applications:

- Enable TLS between the application and the cluster, and between cluster members. Set ssl-components-enabled=all and ssl-endpoint-identification-enabled=true.
- Implement and configure a security manager that can be adapted to your application's security needs. For example, your security manager can establish Role-based Access Control (RBAC) and define a set of authorized resource permissions granted for users in those roles. See Security Implementation Introduction and Overview and Custom Security Manager Example for details.
- Establish Firewall rules as needed to secure inbound and outbound communications for cluster members.

- If using persistent regions, enable disk volume encryption in the Operating System. Consult your system's OS documentation for implementation details.
- Client applications are part of the trust boundary. Validate user input data and objects before writing to GemFire. See the OWASP Secure Coding Practices Input Validation Checklist for a list of recommended practices.

If you wish to further increase your security, then set validate-serializable-objects=true to prevent deserialization of arbitrary Java objects within GemFire and add your classes to serializable-object-filter.

Using OQL Method Authorizers

When the SecurityManager is enabled, by default GemFire throws a NotAuthorizedException when a method within a query is invoked and does not belong to the list of default allowed methods. The MethodInvocationAuthorizer is used to determine whether a specific method invocation on a given object should be allowed or denied during the execution of a particular OQL query.

The default restrictions are strict but customizable. You can tighten or relax the default restrictions under particular circumstances using OQL method authorizers to determine whether a specific method invocation on a given object should be allowed or denied during the execution of an OQL query.

Communication Ports

GemFire processes use TCP/IP ports to communicate with other processes or clients.

For example:

- Clients connect to a locator to discover cache servers.
- JMX clients (such as gfsh and JConsole) can connect to JMX Managers and other manageable 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 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.

Resources That Must Be Protected

These configuration files should be readable and writeable *only* by the dedicated user who runs servers:

- gemfire.properties
- cache.xml
- gfsecurity.properties A default gfsecurity.properties is not provided in the defaultConfigs directory. If you choose to use this properties file, you must create it manually. A clear text user name and associated clear text password may be in this file for authentication purposes. The file system's access rights are relied upon to protect this sensitive information.

The default location of the gemfire.properties and cache.xml configuration files is the defaultConfigs child directory of the main 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 for more details.

The 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-%u_%g.log: Contains logging information of an individual gfsh environment and session.

> By default, gfsh session logging is activated. To enable gfsh logging, you must set the Java system property -Dgfsh. log-level=desired_log_level. See Configuring the gfsh Environment for more information.

These log files should be readable and writable *only* by the dedicated user that runs the servers.

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.

Enable Security with Property Definitions

This topic explains how to enable security with property definitions in VMware Tanzu GemFire.

security-manager Property

The authentication callback and the authorization callback that implement the SecurityManager interface are specified with the security-manager property. When this property is defined, authentication and authorization are enabled. The definition of the security-manager property is the fully qualified name of the class that implements the SecurityManager interface.

For example:

```
security-manager = com.example.security.MySecurityManager
```

Apply security-manager to All Members

To ensure that the security-manager property is applied consistently across a cluster, follow these guidelines:

- Specify the security-manager property in a properties file, such as gemfire.properties, not in a cluster configuration file (such as cluster.properties).
- Specify the properties file when you start the first locator for the cluster.

Is Cluster Management Enabled?

The next steps in applying the security-manager property across the cluster depend on whether cluster management is enabled. Cluster management is enabled when two conditions are met:

- Every locator in the cluster sets --enable-cluster-configuration=true.
- Every server in the cluster sets --use-cluster-configuration=true.

These are the default settings, so unless you have changed them, cluster management is probably enabled for your system, but be sure and confirm before proceeding. Some systems that implement cluster management for most members might include a few servers that do not participate (for which --use-cluster-configuration=false). See Using the Cluster Configuration Service for details.

Apply security-manager to Non-participating Servers

- If cluster management is enabled (the default), the locator will propagate the security-manager setting to all members (locators and servers) that are subsequently started.
- If cluster management is enabled but some servers do not participate in cluster management (that is, servers for which --use-cluster-configuration=false), you must specify the security-manager property for those non-participating servers. Make sure its value is exactly identical to that specified for the first locator.
- If cluster management is not enabled, you must specify the security-manager property for all servers. Make sure its value is exactly identical to that specified for the first locator.

Callbacks

All components of the system invoke the same callbacks. Here are descriptions of the components and the connections that they make with the system.

- A client connects with a server and makes operation requests of that server. The callbacks invoked are those defined by the SecurityManager interface for that server.
- A server connects with a locator, invoking the authenticate callback defined for that locator.
- Components communicating with a locator's JMX manager connect and make operation requests of the locator. The callbacks invoked are those defined by the SecurityManager interface for that locator. Both gfsh and Pulse use this form of communication.
- Applications communicating via the REST API make of a server invoke security callbacks upon connection and operation requests.
- Requests that a gateway sender makes of a locator invoke security callbacks defined for that locator.

security-post-processor Property

The PostProcessor interface allows the definition of a set of callbacks that are invoked after operations that get data, but before the data is returned. This permits the callback to intervene and format the data that is to be returned. The callbacks do not modify the region data, only the data to be returned.

Enable the post processing of data by defining the security-post-processor property with the path to the definition of the interface. For example,

```
security-post-processor = com.example.security.MySecurityPostProcessing
```

Authentication

This topic links to documentation about security authentication in VMware Tanzu GemFire.

Authentication verifies the identities of components within the cluster such as peers, clients, and those connecting to a JMX manager.

- Implementing Authentication

  All components of the cluster authenticate the same way, through a custom-written method.

- Authentication Example

  The example demonstrates the basics of an implementation of the SecurityManager.authenticate method.

- Implementing Authentication Expiry

  Client credentials can be given a limited life time to enhance the security of the cluster.

Implementing Authentication

This topic explains how authentication works in VMware Tanzu GemFire and how to implement it.

Authentication lends a measure of security to a cluster by verifying the identity of components as they connect to the system. All components use the same authentication mechanism.

How Authentication Works

When a component initiates a connection to the cluster, the SecurityManager.authenticate method is invoked. The component provides its credentials in the form of properties as a parameter to the authenticate method.

The credentials parameter is generated by the security-client-auth-init class's getCredentials() call, for example a token, a certificate, or a user/password combination. The authenticate method is expected to either return an object representing a principal or throw an AuthenticationFailedException or AuthenticationExpiredException. The principal object is what is passed on to the authorize method which is discussed in detail in the authorization section.

In case of an AuthenticationExpiredException the Tanzu GemFire client code will make one automatic attempt to re-connect to the member that sent the exception. A SecurityManager implementation that supports re-authentication using expiring credentials must also support non-expiring credentials for cluster members.

A well-designed authenticate method will have a set of known credentials, such as user and password pairs, that can be compared to the credentials presented or will have a way of obtaining those credentials.

How a Server Sets Its Credentials

In order to connect with a locator that performs authentication, a server must set its credentials, a username and password specified as the two properties security-username and security-password. In case of systems that use tokens for authentication the property security-token should be provided.

Choose one of two ways to set the server credentials:

- Add settings to the server properties file, if the credentials comprise a simple username/password combination, or
- Implement the AuthInitialize interface for the server

Add Settings to the Server Properties File

Set security-username and security-password or security-token in the server's gfsecurity.properties file, which is read upon server startup. For example:

```
security-username=admin
security-password=xyz1234
```

Or:

```
security-token=abcdxyz
```

The username, password, and tokens are generally base64 encoded strings which are stored in cleartext, so the gfsecurity.properties file must be protected by restricting access with file system permissions.

Implement the AuthInitialize Interface for the Server

To implement the AuthInitialize interface for the server, set the security-peer-auth-init property so that an object of the class that implements the AuthInitialize interface will be instantiated. There are two ways to do this:

- You can set the security-peer-auth-init property to the fully-qualified class name that implements the AuthInitialize interface as in the example

  ```
  security-peer-auth-init=com.example.security.ServerAuthenticate
  ```

- You can set the security-peer-auth-init property to the fully-qualified method name of a method that instantiates an object of the class that implements the AuthInitialize interface as in the example

  ```
  security-peer-auth-init=com.example.security.ServerAuthenticate.create
  ```

Implement the getCredentials method within the AuthInitialize interface to acquire values for the security-token property or the security-username and security-password properties in whatever way you wish. For example, it might look up values in a database or another external resource.

Gateway senders and receivers communicate as components of their respective server members. Therefore, the credentials of the server become those of the gateway sender or receiver.

How a Client Cache Sets its Credentials

In order to connect with a locator or a server that performs authentication, a client must set its credentials. The credentials parameter is generated by the security-client-auth-init class's getCredentials() call, for example a token, a certificate, or a user/password combination.

You must perform two actions to set to set the client credentials:

- Implement the AuthInitialize interface for the client
- Provide Authinitialize.getCredentials() with secure access to the client credentials

Implement the AuthInitialize Interface for the Client

To implement the AuthInitialize interface for the client, set the security-client-auth-init property, so that an object of the class that implements the AuthInitialize interface will be instantiated. There are two ways to do this:

- You can set the security-client-auth-init property to the fully-qualified class name that implements the AuthInitialize interface as in the example:

  ```
  security-client-auth-init=com.example.security.ClientAuthInitialize
  ```

- You can set the security-client-auth-init property to the fully-qualified name of a static method that instantiates an object of the class that implements the AuthInitialize interface as in the example

  ```
  security-client-auth-init=com.example.security.ClientAuthInitialize.create
  ```

Implement the getCredentials() method of the AuthInitialize interface for the client to acquire values for the security-token property or the security-username and security-password properties in whatever way wish. For example, it might look up values in a database or another external resource, or it might prompt for values.

When implementing the getCredentials() method for a token based system keep in mind that the token provider may return an existing token for a user and that this token may expire sooner than expected. Make sure to understand the implications of this and consider building in a check for imminent expiry in the getCredentials() implementation so that a newly fetched but soon to be expired token does not cause undesired exceptions when used for operations.

Provide Secure Access to Client Credentials

Set the security-token property or the security-username and security-password properties for the client in a way that can be accessed by the getCredentials implementation in AuthInitialize. This can be done via the APIs, properties file or other external sources:

```
Properties properties = new Properties();
properties.setProperty("security-username", "exampleuser23");
properties.setProperty("security-password", "xyz1234");
ClientCache cache = new ClientCacheFactory(properties).create();
```

For security, take care that credentials set in this manner are not accessible to observers of the code.

How Other Components Set Their Credentials

gfsh prompts for the username and password upon invocation of a gfsh connect command. These username/password combinations will be provided as properties to the authenticate method in the keys of security-username and security-password.

Pulse prompts for the username and password upon start up.

Due to the stateless nature of the REST API, a web application or other component that speaks to a server or locator via the REST API goes through authentication on each request. The header of the request needs to include attributes that define values for security-username and security-password or in case of token based security the appropriate header associated with the scheme such as Authorization: Bearer [encoded token-string] for OAuth

Implement SecurityManager Interface

Complete these items to implement authentication done by either a locator or a server.

- Decide upon an authentication algorithm. The Authentication Example stores a set of username and password pairs that represent the identities of components that will connect to the system. This simplistic algorithm returns the username as a principal if the username and password passed to the authenticate method are a match for one of the stored pairs.
- Define the security-manager property. See Enable Security with Property Definitions for details about this property.
- Implement the authenticate method of the SecurityManager interface.
- Define any extra resources that the implemented authentication algorithm needs in order to make a decision.

Authentication Example

This topic presents a VMware Tanzu GemFire authorization example.

This example demonstrates the basics of an implementation of the SecurityManager.authenticate method. The remainder of the example may be found in the VMware Tanzu GemFire source code in the gemfire-core/src/main/java/org/apache/geode/examples/security directory.

The security implementation of every installation is unique, so this example cannot be used in a production environment. Its use of the user name as a returned principal upon successful authentication is a particularly poor design choice, as any attacker that discovers the implementation can potentially spoof the system.

This example assumes that a set of user name and password pairs representing users that may be successfully authenticated has been read into a data structure upon initialization. Any component that presents the correct password for a user name successfully authenticates, and its identity is verified as that user. Therefore, the implementation of the authenticate method checks that the user name provided within the credentials parameter is in its data structure. If the user name is present, then the password provided within the credentials parameter is compared to the data structure's known password for that user name. Upon a match, the authentication is successful.

Username and Password Authentication Example

```
public Object authenticate(final Properties credentials)
    throws AuthenticationFailedException {
    String user = credentials.getProperty(ResourceConstants.USER_NAME);
    String password = credentials.getProperty(ResourceConstants.PASSWORD);

    User userObj = this.userNameToUser.get(user);
    if (userObj == null) {
        throw new AuthenticationFailedException(
            "SampleSecurityManager: wrong username/password");
    }

    if (user != null
        && !userObj.password.equals(password)
        && !"".equals(user)) {
        throw new AuthenticationFailedException(
            "SampleSecurityManager: wrong username/password");
    }

    return user;
}
```

Token with Expiry Authentication Example

```
public Object authenticate(final Properties credentials)
    throws AuthenticationFailedException, AuthenticationExpiredException {
    String encodedToken = credentials.getProperty(ResourceConstants.TOKEN);

    if (this.tokenHasExpired(encodedToken)) {
        throw new AuthenticationExpiredException("SampleSecurityManager: token has expired");
    }
    User user = this.encodedTokenToUser.get(encodedToken);
    if (user == null) {
        throw new AuthenticationFailedException("SampleSecurityManager: unable to get user from token");
    }

    return user;
}
```

Implementing Authentication Expiry

This topic explains how to implement authentication expiry in VMware Tanzu GemFire.

Authentication expiry is supported only with client connections. The use of expirable credentials is most common when used in combination with token-based authentication and authorization. Authentication expiry makes it possible for cluster administrators to limit the life span of client connections within the cluster.

Client connections are notified of expiry by AuthenticationExpiredException, which is thrown in the implementations of SecurityManager.authenticate or SecurityManager.authorize.

Upon receiving the AuthenticationExpiredException, clients will make one automatic attempt to gather new credentials and reconnect (AuthInitialize.getCredentials()). Upon receiving a second AuthenticationExpiredException the exception is thrown back to the user to handle. There are some differences in automatic reconnect behavior between older and newer clients. The following table indicates the client versions that support automatic reconnect.

Client version	single user ops	multi user ops	single user CQ/RI	multi user CQ/RI
Tanzu GemFire 9.15 and later	Y	Y	Y	N
Older than Tanzu GemFire 9.15	Y	Y	N	N

Clients older than Tanzu GemFire v9.15 are also able to do an automatic reconnect unless the connection is one of the following types where the exception will always be propagated up the chain:

- multi-user client mode
- event-dispatching (CQ and registered interest)

Authentication Expiry Considerations

The common cycle for authentication and authorization is the following:

```
AuthInitialize.getCredentials(...) -> SecurityManager.authenticate(...) -> SecurityManager.authorize(...)
```

In case of the use of an external token provider we assume that this token provider will be asked for a token in the AuthInitialize.getCredentials() call. A token provider can return existing tokens (which are about to expire) for a given user, so it is recommended that implementers of the AuthInitialize and SecurityManager

interfaces take imminent timeout and token refresh in consideration to avoid receiving multiple unintended AuthenticationExpiredExceptions in a row and having to deal with the propagation of these exceptions.

Authorization

This topic links to documentation about security authorization in VMware Tanzu GemFire.

Cluster and cache operations can be restricted, intercepted and modified, or completely blocked based on configured access rights set for the various cluster components.

- Implementing Authorization

  To use authorization for client/server systems, your client connections must be authenticated by their servers.

- Method Invocation Authorizers

  Authorizers determine whether a specific method invocation on a given object should be allowed or denied during the execution of a particular OQL query.

- Authorization Examples

  This topic discusses the authorization examples provided in the product under gemfire-core/src/main/java/org/apache/geode/examples/security.

Implementing Authorization

This topic explains how authorization works in VMware Tanzu GemFire and how to implement it.

How Authorization Works

When a component requests an operation, the SecurityManager.authorize method is invoked. It is passed the principal of the operation's requester and a ResourcePermission, which describes the operation requested.

The implementation of the SecurityManager.authorize method makes a decision as to whether the principal will be granted permission to carry out the operation. It returns a boolean in which a return value of true permits the operation, and a return value of false prevents the operation. The operation can also throw an AuthenticationExpiredException.

In case of an AuthenticationExpiredException the Tanzu GemFire client code will make one automatic attempt to re-connect to the member that sent the exception.

A well-designed authorize method will have or will have a way of obtaining a mapping of principals to the operations (in the form of resource permissions) that they are permitted to do.

Resource Permissions

All operations are described by an instance of the ResourcePermission class. A permission contains the Resource data member, which classifies whether the operation as working on

- cache data; value is DATA
- the cluster; value is CLUSTER

A permission also contains the Operation data member, which classifies whether the operation as

- reading; value is READ
- changing information; value is WRITE

- making administrative changes; value is MANAGE

The operations are not hierarchical; MANAGE does not imply WRITE, and WRITE does not imply READ.

Some DATA operations further specify a region name in the permission. This permits restricting operations on that region to only those authorized principals. And within a region, some operations may specify a key. This permits restricting operations on that key within that region to only those authorized principals.

Some CLUSTER operations further specify a finer-grained target for the operation. Specify the target with a string value of:

- DISK to target operations that write to a disk store
- GATEWAY to target operations that manage gateway senders and receivers
- QUERY to target operations that manage both indexes and continuous queries
- DEPLOY to target operations that deploy code to servers
- LUCENE to target Lucene index operations

This table classifies the permissions assigned for operations common to a Client-Server interaction.

Client Operation	Assigned ResourcePermission
get function attribute	CLUSTER:READ
create region	DATA:MANAGE
destroy region	DATA:MANAGE
Region.Keyset	DATA:READ:RegionName
Region.query	DATA:READ:RegionName
Region.getAll	DATA:READ:RegionName
Region.getAll with a list of keys	DATA:READ:RegionName:Key
Region.getEntry	DATA:READ:RegionName
Region.containsKeyOnServer(key)	DATA:READ:RegionName:Key
Region.get(key)	DATA:READ:RegionName:Key
Region.registerInterest(key)	DATA:READ:RegionName:Key
Region.registerInterest(regex)	DATA:READ:RegionName
Region.unregisterInterest(key)	DATA:READ:RegionName:Key
Region.unregisterInterest(regex)	DATA:READ:RegionName
execute function	Defaults to DATA:WRITE. Override Function.getRequiredPermissions to change the permission.
clear region	DATA:WRITE:RegionName
Region.putAll	DATA:WRITE:RegionName
Region.clear	DATA:WRITE:RegionName
Region.removeAll	DATA:WRITE:RegionName
Region.destroy(key)	DATA:WRITE:RegionName:Key
Region.invalidate(key)	DATA:WRITE:RegionName:Key

Client Operation	Assigned ResourcePermission
Region.destroy(key)	DATA:WRITE:RegionName:Key
Region.put(key)	DATA:WRITE:RegionName:Key
Region.replace	DATA:WRITE:RegionName:Key
queryService.newCq	DATA:READ:RegionName
CqQuery.stop	CLUSTER:MANAGE:QUERY

This table classifies the permissions assigned for gfsh operations.

gfsh Command	Assigned ResourcePermission
alter async-event-queue	CLUSTER:MANAGE:DEPLOY
alter disk-store	CLUSTER:MANAGE:DISK
alter query-service	CLUSTER:MANAGE
alter region	DATA:MANAGE:RegionName
alter runtime	CLUSTER:MANAGE
backup disk-store	DATA:READ and CLUSTER:WRITE:DISK
change loglevel	CLUSTER:WRITE
clear defined indexes	CLUSTER:MANAGE:QUERY
close durable-client	CLUSTER:MANAGE:QUERY
close durable-cq	CLUSTER:MANAGE:QUERY
compact disk-store	CLUSTER:MANAGE:DISK
configure pdx	CLUSTER:MANAGE
create async-event-queue	CLUSTER:MANAGE:DEPLOY, plus CLUSTER:WRITE:DISK if the associated region is persistent
create defined indexes	CLUSTER:MANAGE:QUERY
create disk-store	CLUSTER:MANAGE:DISK
create gateway-receiver	CLUSTER:MANAGE:GATEWAY
create gateway-sender	CLUSTER:MANAGE:GATEWAY
create index	CLUSTER:MANAGE:QUERY
create jndi-binding	CLUSTER:MANAGE
create lucene index	CLUSTER:MANAGE:LUCENE
create region	DATA:MANAGE, plus CLUSTER:WRITE:DISK if the associated region is persistent
define index	CLUSTER:MANAGE:QUERY
deploy	CLUSTER:MANAGE:DEPLOY
describe client	CLUSTER:READ
describe config	CLUSTER:READ
describe disk-store	CLUSTER:READ

gfsh Command	Assigned ResourcePermission
describe jndi-binding	CLUSTER:READ
describe lucene index	CLUSTER:READ:LUCENE
describe member	CLUSTER:READ
describe offline-disk-store	CLUSTER:READ
describe query-service	CLUSTER:READ
describe region	CLUSTER:READ
destroy async-event-queue	CLUSTER:MANAGE
destroy disk-store	CLUSTER:MANAGE:DISK
destroy function	CLUSTER:MANAGE:DEPLOY
destroy index	CLUSTER:MANAGE:QUERY
destroy jndi-binding	CLUSTER:MANAGE
destroy lucene index	CLUSTER:MANAGE:LUCENE
destroy region	DATA:MANAGE
execute function	Defaults to DATA:WRITE. Override Function.getRequiredPermissions to change the permission.
export cluster-configuration	CLUSTER:READ
export config	CLUSTER:READ
export data	CLUSTER:READ
export logs	CLUSTER:READ
export offline-disk-store	CLUSTER:READ
export stack-traces	CLUSTER:READ
gc	CLUSTER:MANAGE
get -key=key1 -region=region1	DATA:READ:RegionName:Key
import data	DATA:WRITE:RegionName
import cluster-configuration	CLUSTER:MANAGE
list async-event-queues	CLUSTER:READ
list clients	CLUSTER:READ
list deployed	CLUSTER:READ
list disk-stores	CLUSTER:READ
list durable-cqs	CLUSTER:READ
list functions	CLUSTER:READ
list gateways	CLUSTER:READ
list indexes	CLUSTER:READ:QUERY
list jndi-binding	CLUSTER:READ

gfsh Command	Assigned ResourcePermission
list lucene indexes	CLUSTER:READ:LUCENE
list members	CLUSTER:READ
list regions	CLUSTER:READ
load-balance gateway-sender	CLUSTER:MANAGE:GATEWAY
locate entry	DATA:READ:RegionName:Key
netstat	CLUSTER:READ
pause gateway-sender	CLUSTER:MANAGE:GATEWAY
put --key=key1 --region=region1	DATA:WRITE:RegionName:Key
query	DATA:READ:RegionName
rebalance	DATA:MANAGE
remove	DATA:WRITE:RegionName or DATA:WRITE:RegionName:Key
resume async-event-queue-dispatcher	CLUSTER:MANAGE
resume gateway-sender	CLUSTER:MANAGE:GATEWAY
revoke mising-disk-store	CLUSTER:MANAGE:DISK
search lucene	DATA:READ:RegionName
show dead-locks	CLUSTER:READ
show log	CLUSTER:READ
show metrics	CLUSTER:READ
show missing-disk-stores	CLUSTER:READ
show statistics	CLUSTER:READ
show subscription-queue-size	CLUSTER:READ
shutdown	CLUSTER:MANAGE
start gateway-receiver	CLUSTER:MANAGE:GATEWAY
start gateway-sender	CLUSTER:MANAGE:GATEWAY
start server	CLUSTER:MANAGE
status cluster-config-service	CLUSTER:READ
status gateway-receiver	CLUSTER:READ
status gateway-sender	CLUSTER:READ
status locator	CLUSTER:READ
status server	CLUSTER:READ
stop gateway-receiver	CLUSTER:MANAGE:GATEWAY
stop gateway-receiver	CLUSTER:MANAGE:GATEWAY
stop locator	CLUSTER:MANAGE

gfsh Command	Assigned ResourcePermission
stop server	CLUSTER:MANAGE
undeploy	CLUSTER:MANAGE:DEPLOY

The gfsh connect does not have a permission, as it is the operation that invokes authentication. These gfsh commands do not have permission defined, as they do not interact with the cluster:

- gfsh describe connection, which describes the gfsh end of the connection
- gfsh debug, which toggles the mode within gfsh
- gfsh exit
- gfsh help
- gfsh hint
- gfsh history
- gfsh run, although individual commands within the script will go through authorization
- gfsh set variable
- gfsh sh
- gfsh sleep
- validate offline-disk-store
- gfsh version

This table classifies the permissions assigned for JMX operations.

JMX Operation	Assigned ResourcePermission
DistributedSystemMXBean.shutdownAllMembers	CLUSTER:MANAGE
ManagerMXBean.start	CLUSTER:MANAGE
ManagerMXBean.stop	CLUSTER:MANAGE
ManagerMXBean.createManager	CLUSTER:MANAGE
ManagerMXBean.shutDownMember	CLUSTER:MANAGE
Mbeans get attributes	CLUSTER:READ
MemberMXBean.showLog	CLUSTER:READ
DistributedSystemMXBean.changerAlertLevel	CLUSTER:WRITE
ManagerMXBean.setPulseURL	CLUSTER:WRITE
ManagerMXBean.setStatusMessage	CLUSTER:WRITE
CacheServerMXBean.closeAllContinuousQuery	CLUSTER:MANAGE:QUERY
CacheServerMXBean.closeContinuousQuery	CLUSTER:MANAGE:QUERY
CacheServerMXBean.executeContinuousQuery	DATA:READ
CqQuery.execute	DATA:READ:RegionName and CLUSTER:MANAGE:QUERY
CqQuery.executeWithInitialResults	DATA:READ:RegionName and CLUSTER:MANAGE:QUERY
DiskStoreMXBean.flush	CLUSTER:MANAGE:DISK

JMX Operation	Assigned ResourcePermission
DiskStoreMXBean.forceCompaction	CLUSTER:MANAGE:DISK
DiskStoreMXBean.forceRoll	CLUSTER:MANAGE:DISK
DiskStoreMXBean.setDiskUsageCriticalPercentage	CLUSTER:MANAGE:DISK
DiskStoreMXBean.setDiskUsageWarningPercentage	CLUSTER:MANAGE:DISK
DistributedSystemMXBean.revokeMissingDiskStores	CLUSTER:MANAGE:DISK
DistributedSystemMXBean.setQueryCollectionsDepth	CLUSTER:MANAGE:QUERY
DistributedSystemMXBean.setQueryResultSetLimit	CLUSTER:MANAGE:QUERY
DistributedSystemMXBean.backupAllMembers	DATA:READ and CLUSTER:WRITE:DISK
DistributedSystemMXBean.queryData	DATA:READ
DistributedSystemMXBean.queryDataForCompressedResult	DATA:READ
GatewayReceiverMXBean.pause	CLUSTER:MANAGE:GATEWAY
GatewayReceiverMXBean.rebalance	CLUSTER:MANAGE:GATEWAY
GatewayReceiverMXBean.resume	CLUSTER:MANAGE:GATEWAY
GatewayReceiverMXBean.start	CLUSTER:MANAGE:GATEWAY
GatewayReceiverMXBean.stop	CLUSTER:MANAGE:GATEWAY
GatewaySenderMXBean.pause	CLUSTER:MANAGE:GATEWAY
GatewaySenderMXBean.rebalance	CLUSTER:MANAGE:GATEWAY
GatewaySenderMXBean.resume	CLUSTER:MANAGE:GATEWAY
GatewaySenderMXBean.start	CLUSTER:MANAGE:GATEWAY
GatewaySenderMXBean.stop	CLUSTER:MANAGE:GATEWAY
LockServiceMXBean.becomeLockGrantor	CLUSTER:MANAGE
MemberMXBean.compactAllDiskStores	CLUSTER:MANAGE:DISK

Implement Authorization

Complete these items to implement authorization.

- Decide upon an authorization algorithm. The Authorization Example stores a mapping of which principals (users) are permitted to do which operations. The algorithm bases its decision on a look up of the permissions granted to the principal attempting the operation.
- Define the security-manager property. See Enable Security with Property Definitions for details about this property.
- Implement the authorize method of the SecurityManager interface.
- Define any extra resources that the implemented authorization algorithm needs in order to make a decision.

Authorization of Function Execution

By default, a function executed on servers requires that the entity invoking the function have DATA:WRITE permission on the regions involved. Since the default permission may not be appropriate for all functions, the permissions required may be altered.

To implement a different set of permissions, override the Function.getRequiredPermissions() method in the function's class. The method should return a Collection of the permissions required of the entity that invokes an execution of the function.

Authorization of Methods Invoked from Queries

Enabling the SecurityManager affects queries by restricting the methods that a running query may invoke. See Method Invocations and Method Invocation Authorizers for details.

Method Invocation Authorizers

This topic explains how to use method invocation authorizers in VMware Tanzu GemFire.

Overview

When the SecurityManager is enabled, by default Tanzu GemFire throws a NotAuthorizedException when a method within a query is invoked and does not belong to the list of default allowed methods, given in RestrictedMethodAuthorizer.

The MethodInvocationAuthorizer is used to determine whether a specific method invocation on a given object should be allowed or denied during the execution of a particular OQL query.

Allowing users to execute arbitrary methods on any object present within the Tanzu GemFire member's classpath could impact the integrity of the data and the system on which Tanzu GemFire is running. To avoid this problem, it is always recommended to enable a SecurityManager at the cluster level, give users only the permissions they require, and configure a MethodInvocationAuthorizer that meets your needs.

The main threats to which a Tanzu GemFire cluster might be exposed without a MethodInvocationAuthorizer are highlighted below.

Java Reflection

Allows the user to do anything within the JVM.

```
SELECT * FROM /region r WHERE r.getClass().forName('java.lang.Runtime').getDeclaredMethods()[0].invoke()
```

Cache Modification

Allows the user to do anything with the Cache: close it, access other regions, etc.

```
SELECT * FROM /region.getCache().close()
```

Region Modification

Allows the user to destroy, add or invalidate the entire Region, or specific entries.

```
SELECT * FROM /region.destroyRegion()
SELECT * FROM /region.put('xyz','abc')
SELECT * FROM /region.invalidate('xyz')
```

Region Modification

Allows the user to mutate the state of specific entries.

```
SELECT r.setName('newName') FROM /region r
```

GemFire Authorizers

Tanzu GemFire provides four authorizers by default, each one designed and implemented for a specific use case in mind. VMware recommends that you always use one of these authorizers, and only implement your own if your use case needs are not already met by one of them.

All of the implementations provided by GemFire are designed to prevent security problems and have been thoroughly tested. Extra care should be taken, however, when configuring the internals of some of the authorizers as an incorrect configuration might introduce security holes into the system.

The table below shows a summary of which security threats are fully addressed by each authorizer and which ones might be exploitable, depending on how they are configured (details are shown later for each implementation).

Authorizer/Threat	Reflection	Cache Access	Region Access	Entry Modification
RestrictedMethodAuthorizer	Yes	Yes	Yes	Yes
UnrestrictedMethodAuthorizer	Yes	Yes	Yes	No
RegExMethodAuthorizer	Yes	Yes	Yes	No
JavaBeanAccessorMethodAuthorizer	Yes	Yes	Yes	No

RestrictedMethodAuthorizer

The default MethodInvocationAuthorizer used by GemFire to determine whether a method is allowed to be executed on a specific object instance or not.

The implementation forbids the invocation of all methods during a query execution, except for the ones shown below:

Class	Allowed Methods
java.lang.Object	equals, toString, compareTo
java.lang.Boolean	booleanValue
java.lang.Number	byteValue, intValue, doubleValue, floatValue, longValue, shortValue
java.util.Date	after, before, getTime
java.sql.Timestamp	getNanos
java.lang.String	chartAt, codePointAt, codePointBefore, codePointCount, compareToIgnoreCase, concat, contains, contentEquals, endsWith, equalsIgnoreCase, getBytes, hashCode, indexOf, intern, isEmpty, lastIndexOf, length, matches, offsetByCodePoints, replace, replaceAll, replaceFirst, split, startsWith, substring, toCharArray, toLowerCase, toUpperCase, trim
java.util.Map.Entry, org.apache.geode.cache.Region.Entry	getKey, getValue
java.util.Collection, java.util.Map, org.apache.geode.cache.Region	get, entrySet, keySet, values, getEntries, getValues, containsKey

The authorizer also provides utilities that can be used by custom implementations to determine whether a method is permanently forbidden or, if the method belongs to GemFire, whether it is considered safe to be used within a query execution.

The methods getClass, readObject, readResolve, readObjectNoData, writeObject and writeReplace are permanently forbidden.

The below table shows those methods that belong to GemFire and are considered safe (for methods on org.apache.geode.cache.Region, the authorizer also verifies that the user has the DATA:READ:RegionName permission).

Class	Allowed Methods
org.apache.geode.cache.Region.Entry	getKey, getValue
org.apache.geode.cache.Region	get, entrySet, keySet, values, getEntries, getValues, containsKey

UnrestrictedMethodAuthorizer

A less restrictive MethodInvocationAuthorizer that allows any method invocation during the query execution as long as the following conditions are met:

- The method is not considered permanently forbidden by the RestrictedMethodAuthorizer.
- The method does not belong to GemFire, or does belong but is considered safe by the RestrictedMethodAuthorizer.

This authorizer implementation addresses only three of the four main security risks: Java Reflection, Cache Modification and Region Modification. The Region Entry Modification security risk still exists: users with the DATA:READ:RegionName permission will be able to execute ANY method (even those that mutate the object) on the entries stored within the region and on instances used as bind parameters of the query, so this authorizer implementation must be used with extreme care.

> Usage of this authorizer is recommended for secured clusters on which only trusted users and applications have access to the query engine. It might also be used on clusters on which all entries stored are immutable.

JavaBeanAccessorMethodAuthorizer

A more flexible MethodInvocationAuthorizer that allows methods to be invoked during a query execution if and only if all of the following conditions are met:

- The method is not considered permanently forbidden by the RestrictedMethodAuthorizer.
- The method does not belong to GemFire, or does belong but is considered safe by the RestrictedMethodAuthorizer.
- The method follows the design patterns for accessor methods described in the JavaBean Specification 1.01; that is, the method name begins with is or get.
- The target object on which the method will be executed belongs to a set of pre-configured packages.

When used as intended, and assuming that all region entries and bind parameters follow the JavaBean Specification 1.01, this authorizer implementation addresses all four security risks: Java Reflection, Cache Modification, Region Modification and Region Entry Modification. It should be noted, however, that the Region Entry Modification security threat might be re-introduced: users with the DATA:READ:RegionName privilege

will be able to execute any method whose name starts with is or get on the objects stored within the region and on instances used as bind parameters, providing they are in the pre-configured packages. If those methods do not fully follow the JavaBean Specification 1.01 in that accessors do not mutate the object state, then instances could be potentially modified in place.

> Usage of this authorizer is only recommended for secured clusters on which the user has full confidence in that all objects stored within the regions and used as bind parameters follow the JavaBean Specification 1.01. It might also be used on clusters on which all entries stored are immutable.

RegExMethodAuthorizer

A fully flexible MethodInvocationAuthorizer that allows methods to be invoked during the query execution only if the following conditions are met:

- The method is not considered permanently forbidden by the RestrictedMethodAuthorizer.
- The method does not belong to GemFire, or does belong but is considered safe by the RestrictedMethodAuthorizer.
- The fully qualified method name matches at least one of the pre-configured regular expressions.

When correctly configured, this authorizer implementation addresses the four main security risks: Java Reflection, Cache Modification, Region Modification and Region Entry Modification.
For the statement to remain true, however, the regular expressions used must be correctly configured so no mutator methods ever match. If the regular expressions are not restrictive enough, the Region Entry Modification security risk might be potentially re-introduced: users with the DATA:READ:RegionName privilege will be able to execute methods (even those modifying the entry) on the objects stored within the region and on instances used as bind parameters of the query.

> This authorizer must be used with extreme care, it is the most powerful in terms of flexibility and versatility (full control through regular expressions regarding what to allow and what to forbid); but it is also the most dangerous as one small mistake in the configured regular expressions can unexpectedly allow a wide variety of non safe methods to be executed. Usage of this authorizer implementation is recommended only for scenarios in which the user knows exactly what code is deployed to the cluster, allowing a correct configuration of the regular expressions used. It might also be used on clusters on which all entries stored are immutable.

Custom Authorizers

How Authorization Works

It is important to note that the query engine does not have any information about the actual type of the objects while pre-processing or parsing the query itself, neither can it obtain these details before actually executing the query. The actual check to determine whether a method is allowed or not must be executed while the objects are being traversed by the query engine in runtime.

The query engine, however, remembers whether a specific method has been already authorized or not for the current query execution context, meaning that the authorization will be executed only once in the lifetime of a particular query for every new method seen while traversing the objects. Nevertheless, the

authorizer implementation must be highly performant as it will be invoked by GemFire in runtime during the actual query execution.

Implementing a Method Authorizer

Complete these items to implement a custom method authorizer.

- Decide which methods from your domain model should be allowed to be invoked during a query execution.
- Decide which resources, if any, you will need in order to determine whether a method can be invoked or not.
- Implement the initialize method of the MethodInvocationAuthorizer interface to fully configure your implementation, based on the resources needed to execute the authorization.
- Implement the authorize method of the MethodInvocationAuthorizer interface. It must determine whether a method is allowed to be executed on a particular object instance during a query execution. The implementation should be lightning fast and thread safe.

Changing the Method Authorizer

You can set the MethodInvocationAuthorizer to be used by the query engine through the gfsh command-line utility. In addition, you can modify the configured MethodInvocationAuthorizer while members are already running by using the alter query-service command. It is always advisable to make these changes during periods of low activity, though.

The following constraints apply when the MethodInvocationAuthorizer used by the cluster is changed in runtime:

- Queries started after the MethodInvocationAuthorizer is changed will use the newly configured authorizer.
- Queries in flight are not affected. Before the query starts, it picks up the already configured MethodInvocationAuthorizer and will use it until the execution finishes.
- Indexes configured with an expression using methods forbidden by the newly configured MethodInvocationAuthorizer will be marked as invalid the next time a mapping is added or removed from the index.
- Continuous queries already running will pick up the newly configured MethodInvocationAuthorizer the next time the CQ is processed upon the arrival of a cache event. If the CQ has methods forbidden by the newly configured MethodInvocationAuthorizer, any subsequent execution will result in an error during the CQ processing, and the onError method will be invoked for the associated CqListener.

> In order to improve performance, the continuous query engine uses an internal cache to avoid executing the query in scenarios for which the answer can be automatically inferred. These results might become invalid after applying the new security rules, so GemFire deactivates the usage of this optimization until the member is restarted or the query is registered again.

Authorization Example

This topic presents a VMware Tanzu GemFire authorization example.

Disclaimer

The security implementation of every installation is unique. These examples are provided for illustrative purposes only and must not be used in a production environment.

The examples demonstrate the basics for implementing both user authorization (SecurityManager.authorize) and method invocation authorization (MethodInvocationAuthorizer.authorize) during query executions.

User Authorization Example

This example assumes that a set of users, a set of roles that a user might take on within the system, and a mapping of users to their roles are described in a JSON format file. The roles define a set of authorized resource permissions granted for users in those roles. Code not shown here parses the file to compose a data structure with the information about roles and users. The authorize callback denies permission for any operation that does not have a principal representing the identity of the operation's requester. Given the principal, the method iterates through the data structure searching for the necessary permissions for the principal. When the necessary permission is found, authorization is granted by returning the value true. If the permission is not found in the data structure, then the method returns false, denying authorization of the operation.

```
public boolean authorize(final Object principal, final ResourcePermission context) {
    if (principal == null) return false;

    User user = this.userNameToUser.get(principal.toString());
    if (user == null) return false; // this user is not authorized to do anything

    // check if the user has this permission defined in the context
    for (Role role : this.userNameToUser.get(user.name).roles) {
        for (Permission permitted : role.permissions) {
            if (permitted.implies(context)) {
                return true;
            }
        }
    }

    return false;
}
```

Method Invocation Authorization Example

This example assumes that the entire domain model is deployed to the cluster and that the user is allowed to modify these classes. The authorize callback denies access to methods that have been permanently forbidden by the RestrictedMethodAuthorizer and returns false right away. When the method is not permanently forbidden, the implementation checks whether the method has been annotated with a custom annotation. When the necessary annotation is found, authorization is granted by returning the value true. If the annotation is not found, then the method returns false, denying the invocation of the method during the query execution.

```
public boolean authorize(Method method, Object target) {
    // Check if forbidden by default.
    if (defaultAuthorizer.isPermanentlyForbiddenMethod(method, target)) {
        return false;
    }

    // Check if annotation is present
    return method.isAnnotationPresent(Authorized.class);
  }
```

Token with Expiry Authorization Example

This example assumes that the system is using an expirable token that needs to be checked for expiry before resource permissions are verified.

```
public boolean authorize(final Object tokenUser, final ResourcePermission context)
      throws AuthenticationExpiredException {
    // Check expiry
    if (this.tokenHasExpired(tokenUser)) {
        throw new AuthenticationExpiredException("SampleSecurityManager: token has expired");
    }

    // Check permissions
    return this.tokenHasRequiredPermissions(tokenUser, context);
  }
```

Custom Security Manager Example

This topic provides an example that demonstrates the basics of creating a custom VMware Tanzu GemFire SecurityManager with an implementation of the SecurityManager.authenticate and SecurityManager.authorize methods that connect to a Java client.

> The security implementation of every installation is unique. This example should not be used in a production environment. The purpose of this example is to provide a basic example that can be adapted to each user's own security needs.

In this example, you will:

1. Create a list of approved users to authenticate and authorize against.
2. Create a BasicSecurityManager implementation that uses the SecurityManager authenticate and authorize methods.
3. Start a GemFire cluster with the BasicSecurityManager security manager.
4. Create a Java client application that authenticates against the application and PUTS and GETS data into a region.

Create a List of Approved Users

GemFire offers multiple layers of access to a GemFire cluster, which are defined by the GemFire resource permissions. For more information about GemFire resource permissions, see Resource Permissions in *Implementing Authorization*.

In this example, you create a USER class and initialize two users:

- An Operator user that can manage the GemFire cluster but has no data access.
- An Application Developer who can access the GemFire cluster to manager data, but cannot manage the GemFire cluster to perform tasks like deleting the cluster.

Create the USER class as follows.

> You must implement the Serializable interface. This allows GemFire to deserialize the class when checking the client username.

```
import java.io.Serializable;
import java.util.List;

import org.apache.geode.security.ResourcePermission;

public class User implements Serializable {

  List<ResourcePermission> userPermissions;
  String userName;
  String userPassword;

  public User(String userName, String userPassword, List<ResourcePermission> userPermissions) {
    this.userName = userName;
    this.userPassword = userPassword;
    this.userPermissions = userPermissions;
  }

  public String getUserPassword() {
    return userPassword;
  }

  @Override
  public String toString() {
    return userName;
  }

  public List<ResourcePermission> getPermissions() {
    return this.userPermissions;
  }

  public boolean hasPermission(ResourcePermission resourcePermissionRequested) {
    boolean hasPermission = false;

    for (ResourcePermission userPermission : userPermissions) {
      if (userPermission.implies(resourcePermissionRequested)) {
        hasPermission = true;
        break;
      }
    }
    return hasPermission;
  }
}
```

Create a BasicSecurityManager

1. Create a BasicSecurityManager that implements the GemFire SecurityManager interface. In the init method of the BasicSecurityManager two USERS are created: operator and appdeveloper.

> In this example the username and password are hard-coded when creating the Users. This does not represent best practices.

In your application, this list may come from an external system.

In the BasicSecurityManager:

- Create an Operator user with CLUSTER MANAGE, CLUSTER WRITE, and CLUSTER READ permissions. Set the username to “operator” and the password to “secret”.
- Create an Application Developer user with CLUSTER READ, DATA MANAGE, DATA WRITE, DATA READ. Set the username to “appDeveloper” and the password to “NotSoSecret”.
- Add the users to an “approved users” list, which will allow the application to check incoming credentials.

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Properties;

import org.apache.geode.security.AuthenticationFailedException;
import org.apache.geode.security.ResourcePermission;
import org.apache.geode.security.SecurityManager;

public class BasicSecurityManager implements SecurityManager {

  private HashMap<String, User> approvedUsersList = new HashMap<>();

  @Override
  public void init(final Properties securityProperties) {

    List<ResourcePermission> operatorPermissions = new ArrayList<>();
    operatorPermissions.add(new ResourcePermission(ResourcePermission.Resource.CLUSTER,
      ResourcePermission.Operation.MANAGE));
    operatorPermissions.add(new ResourcePermission(ResourcePermission.Resource.CLUSTER,
      ResourcePermission.Operation.WRITE));
    operatorPermissions.add(new ResourcePermission(ResourcePermission.Resource.CLUSTER,
      ResourcePermission.Operation.READ));

    User operator = new User("operator", "secret", operatorPermissions);

    List<ResourcePermission> appDevPermissions = new ArrayList<>();
    appDevPermissions.add(new ResourcePermission(ResourcePermission.Resource.CLUSTER,
      ResourcePermission.Operation.READ));
    appDevPermissions.add(new ResourcePermission(ResourcePermission.Resource.DATA,
      ResourcePermission.Operation.MANAGE));
    appDevPermissions.add(new ResourcePermission(ResourcePermission.Resource.DATA,
      ResourcePermission.Operation.WRITE));
    appDevPermissions.add(new ResourcePermission(ResourcePermission.Resource.DATA,
      ResourcePermission.Operation.READ));

    User appDeveloper = new User("appDeveloper", "NotSoSecret", appDevPermissions);

    this.approvedUsersList.put("operator", operator);
    this.approvedUsersList.put("appDeveloper", appDeveloper);

  }
}
```

2. Implement the authentication and authorization methods.

Authentication

In the authenticate method, you must check the credentials passed into the BasicSecurityManager from the client against those in the approvedUsers list. If the credentials match, the method returns a USER object, authenticatedUser. This is the object is passed into the authorize method when the client application performs an operation.

```
public class BasicSecurityManager implements SecurityManager {

  private HashMap<String, User> approvedUsersList = new HashMap<>();

  @Override
  public void init(final Properties securityProperties) {...}

  @Override
  public Object authenticate(Properties credentials) throws AuthenticationFailedException {

    String usernamePassedIn = credentials.getProperty(USER_NAME);
    String passwordPassedIn = credentials.getProperty(PASSWORD);

    User authenticatedUser = this.approvedUsersList.get(usernamePassedIn);

      if (authenticatedUser == null) {
        throw new AuthenticationFailedException("Wrong username/password");
      }

      if (authenticatedUser != null && !authenticatedUser.getUserPassword().equals(passwordPassedIn)
        && !"".equals(authenticatedUser)) {
          throw new AuthenticationFailedException("Wrong username/password");
      }

    return authenticatedUser;
  }
}
```

Authorization

The object returned from the authenticate method above (the authenticatedUser object from above) is passed into the authorize method as the Object principal. This object is used to authorize the action that the client is attempting to perform.

The resourcePermissionRequested (the action the client wants to perform) is compared with the USERS given permissions that were defined when creating the two users in the init method. If the user is allowed to perform the requested action, then the method returns true. Otherwise, the method returns false and the action is denied.

```
public class BasicSecurityManager implements SecurityManager {

  private HashMap<String, User> approvedUsersList = new HashMap<>();

  @Override
  public void init(final Properties securityProperties) {...}

  @Override
  public Object authenticate(Properties credentials) throws AuthenticationFailedException {...}

  @Override
  public boolean authorize(Object principal, ResourcePermission resourcePermissionRequested) {

    if (principal == null) {
      return false;
    }

    User user = this.approvedUsersList.get(principal.toString());

      if (user == null) {
        return false;
      }

      for (ResourcePermission userPermission : user.getPermissions()) {
        if (userPermission.implies(resourcePermissionRequested)) {
          return true;
        }
      }

    return false;
  }
}
```

Starting a GemFire Cluster with the Security Manager

You must add your BasicSecurityManager to the classpath when starting a GemFire cluster.

1. Build the .jar file for the BasicSecurityManager created above. Record the directory and file path where the jar file is created for use in a later step.
2. On a command line, run gfsh to start GemFire's shell.
3. Start a locator and include the path to the jar file and class name of the BasicSecurityManager. The start locator command will look like this:

   ```
   start locator --name=locator1 --J=-Dgemfire.security-manager=BasicSecurityManager --classpath=/path/to/your/jar/file/BasicSecurityManager-1.0-SNAPSHOT.jar
   ```

   Where: - --J=-Dgemfire.security-manager=BasicSecurityManager defines the package and class for your security manager and allows GemFire to find the class when starting. - --classpath=[path to your jar file]/BasicSecurityManager-1.0-SNAPSHOT.jar defines the path to the jar file that GemFire should use as the security manager.
4. Once the locator has started, you will see output similar to this:

```
Starting a Geode Locator in [path to where GemFire was started] /locator1...
.........
Locator in [path to where GemFire was started]/locator1 on [ip address] [10334] as locator1 is currently online.
Process ID: 75033
Uptime: 11 seconds
Geode Version: 1.15.0-build.0
Java Version: 1.8.0_292
...

Unable to auto-connect (Security Manager may be enabled). Please use "connect --locator=[ip address] [10334] --user --password" to connect Gfsh to the locator.

Authentication required to connect to the Manager.
```

This output shows that locator started. When the security manager is included to start the cluster, it is immediately used to authenticate the current user.

5. Connect to the cluster as the Operator as defined in the BasicSecurityManager class above. This is the only role created for this example that has the authorization to manage the cluster.

   In gfsh, the command would look similar to the following:

   ```
   gfsh: connect --locator= [IP Address that GemFire is running on] [10334] --user=operator --password=secret
   ```

   You should now be connected to the locator.

6. Start a server. This will be very similar to starting the locator. In gfsh, run the start server command, which will include the path to the same BasicSecurityManager.jar file used when starting the locator.

   ```
   gfsh: start server --name=server1 --locators=localhost[10334] --classpath=[path to your security manager]/BasicSecurityManager-1.0-SNAPSHOT.jar --user=operator --password=secret
   ```

   Repeat this step for each server you need to start, but change the server --name= parameter to be unique for each server.

You now have a GemFire cluster running with your BasicSecurityManager.

Create a Java Client application

To create a Java client application that authenticates against the application and PUTS and GETS data into a region:

- You must create a region on the GemFire cluster for the application to interact with.
- The client application must have a class that implements the AuthInitialize interface. This class is used by GemFire to provide the credentials to the cluster. The client application must set its credentials composed of two properties,security username and security-password.
- The client application must set the security-client-auth-init property, which indicates to GemFire the class that implements the AuthInitialize interface.

Create a Region on the GemFire Cluster

Create a region on the GemFire cluster for the application to interact with.

At this point in our example, gfsh is connected as the Operator who has permission to manage the cluster, but does not have permission to interact with or manage data. To create a region, you must disconnect as Operator, connect as appDeveloper, then create the region.

1. In gfsh, disconnect from being the Operator by running:

```
gfsh: disconnect
```

2. In gfsh, connect as the appDeveloper by running:

```
gfsh: connect –-user=appDevleoper –-password=NotSoSecret
```

3. In gfsh, create a region named helloWorld by running the command below. This will create a partitioned region in your GemFire cluster from which you can PUT and GET data. For more information about partitioned regions, see Partitioned Regions.

```
gfsh: create region --name=helloWorld --type=PARTITION
```

Set Credential Properties

You must set the security-username and security-password in the class that implements the AuthInitialize interface. These credentials might come from an external source, such as a credentials database, ActiveDirectory, or some other external system.

> The goal of this example is to demonstrate how the security manager works. To simplify the example, the credentials are hard-coded into the application. This is not a best practice for security.

1. Create the class UserPasswordAuthInit that implements the AuthInitialize interface.

```
import java.util.Properties;
import org.apache.geode.distributed.DistributedMember;
import org.apache.geode.security.AuthInitialize;
import org.apache.geode.security.AuthenticationFailedException;

public class UserPasswordAuthInit implements AuthInitialize {

  @Override
  public Properties getCredentials(Properties properties, DistributedMember distributedMember, boolean isPeer) throw
s AuthenticationFailedException {
    properties.setProperty("security-username", "appDeveloper");
    properties.setProperty("security-password", "NotSoSecret");
    return properties;
  }
}
```

   This basic class sets two properties, security-username and security-password, that match the credentials declared for the appDeveloper user in the BasicSecurityManager class.

2. Set the security-client-auth-init property in the Main class and pass it to the ClientCacheFactory.

```
import java.util.Properties;
import org.apache.geode.cache.Region;
import org.apache.geode.cache.client.ClientCache;
import org.apache.geode.cache.client.ClientCacheFactory;
import org.apache.geode.cache.client.ClientRegionShortcut;

public class Main {

  public static void main (String[] args) {
    Properties properties = new Properties();
    properties.setProperty("security-client-auth-init", UserPasswordAuthInit.class.getName());

    ClientCache cache = new ClientCacheFactory(properties).addPoolLocator("localhost", 10334).create();

    Region<String, String>
    helloWorldRegion = cache.<String, String>createClientRegionFactory(ClientRegionShortcut.PROXY).create("hello
World");

    helloWorldRegion.put("1", "HelloWorldValue");
    String value1 = helloWorldRegion.get("1");
    System.out.println(value1);
    cache.close();
  }
}
```

3. Run the application. You should see output similar to the following:

```
ERROR StatusLogger Log4j2 could not find a logging implementation. Please add log4j-core to the classpath. Using S
impleLogger to log to the console...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

HelloWorldValue

Process finished with exit code 0
```

This outputs the value “HelloWorldValue” that you put in for key 1. If the output includes an authentication error, confirm that you have the correct username and password in your AuthInitialize class. Ignore other ERROR messages for this example.

Client Authorization Error

If you were to remove the appDevelopers permission to WRITE to the GemFire cluster, you would see an error similar to the following:

```
ERROR StatusLogger Log4j2 could not find a logging implementation. Please add log4j-core to the classpath. Using SimpleLogg
er to log to the console...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Exception in thread "main" org.apache.geode.cache.client.ServerOperationException: remote server on [your IP address]: org.ap
ache.geode.security.NotAuthorizedException: appDeveloper not authorized for DATA:WRITE:helloWorld:1
  at org.apache.geode.cache.client.internal.OpExecutorImpl.handleException(OpExecutorImpl.java:554)
  at org.apache.geode.cache.client.internal.OpExecutorImpl.handleException(OpExecutorImpl.java:615)
  at org.apache.geode.cache.client.internal.OpExecutorImpl.handleException(OpExecutorImpl.java:501)
  at org.apache.geode.cache.client.internal.OpExecutorImpl.execute(OpExecutorImpl.java:142)
  at org.apache.geode.cache.client.internal.OpExecutorImpl.execute(OpExecutorImpl.java:108)
  at org.apache.geode.cache.client.internal.PoolImpl.execute(PoolImpl.java:776)
  at org.apache.geode.cache.client.internal.PutOp.execute(PutOp.java:91)
  at org.apache.geode.cache.client.internal.ServerRegionProxy.put(ServerRegionProxy.java:159)
  at org.apache.geode.internal.cache.LocalRegion.serverPut(LocalRegion.java:3048)
  at org.apache.geode.internal.cache.LocalRegion.cacheWriteBeforePut(LocalRegion.java:3165)
  at org.apache.geode.internal.cache.ProxyRegionMap.basicPut(ProxyRegionMap.java:238)
  at org.apache.geode.internal.cache.LocalRegion.virtualPut(LocalRegion.java:5613)
  at org.apache.geode.internal.cache.LocalRegion.virtualPut(LocalRegion.java:5591)
  at org.apache.geode.internal.cache.LocalRegionDataView.putEntry(LocalRegionDataView.java:156)
  at org.apache.geode.internal.cache.LocalRegion.basicPut(LocalRegion.java:5049)
  at org.apache.geode.internal.cache.LocalRegion.validatedPut(LocalRegion.java:1648)
  at org.apache.geode.internal.cache.LocalRegion.put(LocalRegion.java:1635)
  at org.apache.geode.internal.cache.AbstractRegion.put(AbstractRegion.java:442)
  at Main.main(Main.java:21)
Caused by: org.apache.geode.security.NotAuthorizedException: appDeveloper not authorized for DATA:WRITE:helloWorld:1
  at org.apache.geode.internal.security.IntegratedSecurityService.authorize(IntegratedSecurityService.java:292)
  at org.apache.geode.internal.security.IntegratedSecurityService.authorize(IntegratedSecurityService.java:275)
  at org.apache.geode.internal.security.IntegratedSecurityService.authorize(IntegratedSecurityService.java:269)
  at org.apache.geode.internal.cache.tier.sockets.command.Put70.cmdExecute(Put70.java:246)
  at org.apache.geode.internal.cache.tier.sockets.BaseCommand.execute(BaseCommand.java:187)
  at org.apache.geode.internal.cache.tier.sockets.ServerConnection.doNormalMessage(ServerConnection.java:881)
  at org.apache.geode.internal.cache.tier.sockets.ServerConnection.doOneMessage(ServerConnection.java:1070)
  at org.apache.geode.internal.cache.tier.sockets.ServerConnection.run(ServerConnection.java:1344)
  at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
  at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
  at org.apache.geode.internal.cache.tier.sockets.AcceptorImpl.lambda$initializeServerConnectionThreadPool$3(AcceptorImpl.ja
va:690)
  at org.apache.geode.logging.internal.executors.LoggingThreadFactory.lambda$newThread$0(LoggingThreadFactory.java:120)
  at java.lang.Thread.run(Thread.java:748)
Caused by: org.apache.shiro.authz.UnauthorizedException: Subject does not have permission [DATA:WRITE:helloWorld:1]
  at org.apache.shiro.authz.ModularRealmAuthorizer.checkPermission(ModularRealmAuthorizer.java:334)
  at org.apache.shiro.mgt.AuthorizingSecurityManager.checkPermission(AuthorizingSecurityManager.java:141)
  at org.apache.shiro.subject.support.DelegatingSubject.checkPermission(DelegatingSubject.java:214)
  at org.apache.geode.internal.security.IntegratedSecurityService.authorize(IntegratedSecurityService.java:288)
... 12 more

Process finished with exit code 1
```

The above error message points to an org.apache.geode.security.NotAuthorizedException: appDeveloper not authorized for DATA:WRITE:helloWorld:1, showing that the appDeveloper does not have the correct permissions to write to the cluster.

Post Processing of Region Data

This topic explains post processing of region data in VMware Tanzu GemFire.

The PostProcessor interface allows the definition of a callback that is invoked after any and all client and gfsh operations that get data, but before the data is returned. It permits the callback to intervene and

format the data that is to be returned. The callbacks must not modify the region data, only the data to be returned.

The processRegionValue method is given the principal of the operation requester. The operation will already have been completed, implying that the principal will have been authorized to complete the requested operation. The post processing can therefore format the returned data based on the identity of the requester (principal).

By default, the key and value parameters to the processRegionValue method are references to the region entry. Modify copies of these parameters to avoid changing the region entries. Copy on Read Behavior discusses the issue.

The processRegionValue method is invoked for these API calls:

- Region.get
- Region.getAll
- Query.execute
- CqQuery.execute
- CqQuery.executeWithInitialResults
- CqListener.onEvent
- for a relevant region event from CacheListener.afterUpdate for which there is interest registered with Region.registerInterest

Care should be taken when designing a system that implements the post processing callback. It incurs the performance penalty of an extra method invocation on every get operation.

Implement Post Processing

Complete these items to implement post processing.

- Define the security-post-processor property. See Enable Security with Property Definitions for details about this property.
- Implement the processRegionValue method of the PostProcessor interface.

SSL/TLS

This topic describes how SSL/TLS in VMware Tanzu GemFire protects your data in transit between applications by ensuring that only the applications identified by you can share cluster data.

To be secure, the data that is cached in a Tanzu GemFire system must be protected during storage, distribution, and processing. At any time, data in a cluster 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, Tanzu GemFire relies on your standard system security features such as firewalls, operating system settings, and JDK security settings.

The SSL/TLS implementation ensures that only the applications identified by you can share cluster data in transit. In this figure, the data in the visible portion of the cluster 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/TLS for data distribution provides secure communication between Tanzu GemFire system members inside and outside the firewalls.

- Configuring SSL/TLS

  You configure SSL/TLS for mutual authentication between members and to protect your data during distribution. You can use SSL/TLS alone or in conjunction with the other Tanzu GemFire security options.

- SSL/TLS Sample Implementation

  A simple example demonstrates the configuration and startup of Tanzu GemFire system components with SSL/TLS.

Configuring SSL/TLS

This topic explains how to configure SSL for authentication in VMware Tanzu GemFire.

You can configure SSL for authentication between members and to protect your data during distribution. You can use SSL alone or in conjunction with the other Tanzu GemFire security options.

Tanzu GemFire SSL connections use the Java Secure Sockets Extension (JSSE) package, so the properties described here apply to Tanzu GemFire servers and to Java-based clients. SSL configuration in non-Java clients may differ. Refer to your client's documentation for details.

SSL-Configurable Components

You can specify that SSL be used system-wide, or you can independently configure SSL for specific system components. The following list shows the system components that can be separately configured to communicate using SSL, and the kind of communications to which each component name refers:

cluster
: Peer-to-peer communications among members of a cluster

gateway
: Communication across WAN gateways from one site to another

web
: All web-based services hosted on the configured server, which can include the Developer REST API service, the Management REST API service (used for remote cluster management) and the Pulse monitoring tool's web-based user interface.

jmx

Java management extension communications, including communications with the gfsh utility. The Pulse monitoring tool uses JMX for server-side communication with a locator, but SSL applies to this connection only if Pulse is located on an app server separate from the locator. When Pulse and the locator are colocated, JMX communication between the two does not involve a TCP connection, so SSL does not apply.

> Note: GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

locator
: Communication with and between locators

server
: Communication between clients and servers

all
: All of the above (use SSL system-wide)

Specifying that a component is enabled for SSL applies to the component's server-socket side and its client-socket side. For example, if you enable SSL for locators, then any process that communicates with a locator must also have SSL enabled. If you provide "" as the value, SSL is turned off for all components.

SSL Configuration Properties

You can use Tanzu GemFire configuration properties to activate or deactivate SSL, to identify SSL ciphers and protocols, and to provide the location and credentials for key and trust stores.

ssl-enabled-components
: List of components for which to enable SSL. Component list can be "" (deactivate SSL), "all", or a comma-separated list of components.

ssl-endpoint-identification-enabled
: A boolean value that, when set to true, causes clients to validate the server's hostname using the server's certificate. The default value is false. Enabling endpoint identification guards against DNS man-in-the-middle attacks when trusting certificates that are not self-signed.

ssl-use-default-context
: A boolean value that, when set to true, allows Tanzu GemFire to use the default SSL context as returned by SSLContext.getInstance('Default') or set by using SSLContext.setDefault(). When enabled, also causes ssl-endpoint-identification-enabled to be set to true.

ssl-require-authentication
: Requires two-way authentication, applies to all components except web. Boolean - if true (the default), two-way authentication is required.

ssl-web-require-authentication
: Requires two-way authentication for web component. Boolean - if true, two-way authentication is required. Default is false (one-way authentication only).

ssl-default-alias
: A server uses one key store to hold its SSL certificates. All components on that server can share a single certificate, designated by the ssl-default-alias property. If multiple SSL components are enabled and any one of them also specifies a component-specific alias, ssl-default-alias must be set if not all components specify an alias to use.

ssl-*component*-alias=string
: You can configure a separate certificate for any component. All certificates reside in the same key store, but can be designated by separate aliases that incorporate the component name, using this

syntax, where *component* is the name of a component. When a component-specific alias is specified, it overrides the ssl-default-alias for the *component* specified.

For example, ssl-locator-alias would specify a name for the locator component's certificate in the system key store.

ssl-ciphers
: A comma-separated list of the valid ciphers for TCP/IP connections with TLS encryption enabled. A setting of 'any' allows the JSSE provider to select an appropriate cipher that it supports.

ssl-protocols
: A comma-separated list of the valid protocol versions for TCP/IP connections with TLS encryption enabled. A setting of 'any' attempts to use your JSSE provider's TLSv1.3, or TLSv1.2 if v1.3 is not available.

ssl-keystore, ssl-keystore-password
: The path to the key store and the key store password, specified as strings

ssl-truststore, ssl-truststore-password
: The path to the trust store and the trust store password, specified as strings

ssl-keystore-type, ssl-truststore-type
: The types of the key store and trust store, specified as strings. The default for both is "JKS", indicating a Java key store or trust store.

Example: secure communications throughout

To implement secure SSL communications throughout an entire cluster, each process should enable SSL for all components.

```
ssl-enabled-components=all
ssl-endpoint-identification-enabled=true
ssl-keystore=secure/keystore.dat
ssl-keystore-password=changeit
ssl-truststore=secure/truststore.dat
ssl-truststore-password=changeit
```

If the key store has multiple certificates you may want to specify the alias of the one you wish to use for each process. For instance, ssl-default-alias=Hiroki.

Example: non-secure cluster communications, secure client/server

In this example, SSL is used to secure communications between the client and the server:

Server properties

Cluster SSL is not enabled.

```
ssl-enabled-components=server,locator
ssl-server-alias=server
ssl-keystore=secure/keystore.dat
ssl-keystore-password=changeit
ssl-truststore=secure/truststore.dat
ssl-truststore-password=changeit
ssl-default-alias=Server-Cert
```

Locator properties

Cluster SSL is not enabled.

```
ssl-enabled-components=locator
ssl-locator-alias=locator
ssl-keystore=secure/keystore.dat
ssl-keystore-password=changeit
ssl-truststore=secure/truststore.dat
ssl-truststore-password=changeit
ssl-default-alias=Locator-Cert
```

Client properties

On Java clients, the list of enabled components reflects the server's configuration so the client knows how it is expected to communicate with (for example) servers and locators. Paths to keystore and truststore are local to the client.

In this example, the client's trust store must trust both locator and server certificates. Since the client does not specify a certificate alias, SSL will use the default certificate in its key store.

```
ssl-enabled-components=server,locator
ssl-endpoint-identification-enabled=true
ssl-keystore=secret/keystore.dat
ssl-keystore-password=changeit
ssl-truststore=secret/truststore.dat
ssl-truststore-password=changeit
```

SSL Property Reference Tables

The following table lists the components you can configure to use SSL.

Table 1. SSL-Configurable Components

Component	Communication Types
cluster	Peer-to-peer communications among members of a cluster
gateway	Communication across WAN gateways from one site to another
web	Web-based communication, including REST interfaces
jmx	Java management extension communications, including gfsh
locator	Communication with and between locators
server	Communication between clients and servers
all	All of the above

The following table lists the properties you can use to configure SSL on your Tanzu GemFire system.

Table 2. SSL Configuration Properties

Property	Description	Value
ssl-enabled-components	list of components for which to enable SSL	"all", "", or comma-separated list of components: cluster, gateway, web, jmx, locator, server
ssl-endpoint-identification-enabled	causes clients to validate server hostname using server certificate	boolean - if true, does validation; defaults to false
ssl-use-default-context	allows Tanzu GemFire to use the default SSL context	boolean - if true, uses the default SSL context. Also sets ssl-endpoint-identification-enabled to true; defaults to false
ssl-require-authentication	requires two-way authentication, applies to all components except web	boolean - if true (the default), two-way authentication is required

Property	Description	Value
ssl-web-require-authentication	requires two-way authentication for web component	boolean - if true, two-way authentication is required. Default is false (one-way authentication only)
ssl-default-alias	default certificate name	string - if empty, use first certificate in key store
ssl-*component*-alias	component-specific certificate name	string - applies to specified *component*
ssl-ciphers	list of SSL ciphers	comma-separated list (default “any”)
ssl-protocols	list of SSL protocols	comma-separated list (default “any”)
ssl-keystore	path to key store	string
ssl-keystore-password	key store password	string
ssl-keystore-type	trust store type	string
ssl-truststore	path to trust store	string
ssl-truststore-password	trust store password	string
ssl-truststore-type	trust store type	string

Procedure

1. Make sure your Java installation includes the JSSE API and familiarize yourself with its use. For information, see the Oracle JSSE website.
2. Configure SSL as needed for each connection type:
    1. Use locators for member discovery within the clusters and for client discovery of servers. See Configuring Peer-to-Peer Discovery and Configuring a Client/Server System.
    2. Configure SSL properties as necessary for different component types, using the properties described above. For example, to enable SSL for communication between clients and servers you would configure properties in the gemfire.properties file similar to:

    ```
    ssl-enabled-components=server
    ssl-protocols=any
    ssl-ciphers=SSL_RSA_WITH_NULL_MD5, SSL_RSA_WITH_NULL_SHA
    ssl-keystore=/path/to/trusted.keystore
    ssl-keystore-password=password
    ssl-truststore=/path/to/trusted.keystore
    ssl-truststore-password=password
    ```

SSL/TLS Sample Implementation

This topic presents a simple example that demonstrates the configuration and startup of VMware Tanzu GemFire system components with SSL/TLS.

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, run the keytool utility:

```
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/TLS in the gemfire.properties file. In this example, SSL/TLS is enabled for all components.

```
ssl-enabled-components=all
locators=<hostaddress>[<port>]
```

gfsecurity.properties File

You can specify the provider-specific settings in a 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.

```
ssl-keystore=/path/to/trusted.keystore
ssl-keystore-password=password
ssl-truststore=/path/to/trusted.keystore
ssl-truststore-password=password
security-username=xxxx
security-userPassword=yyyy
```

Locator Startup

Before starting other system members, we started the locator with the SSL/TLS 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
```

Connecting to a Running Cluster

You can use gfsh to connect to an SSL/TLS-enabled cluster that is already running by specifying the use-ssl command-line option and providing a path to the security configuration file:

```
gfsh>connect --locator=localhost[10334] --use-ssl \
--security-properties-file=/path/to/your/gfsecurity.properties
```

Once connected, you can then issue gfsh commands to perform a variety of operations, including listing members and displaying region characteristics.

Performance Tuning and Configuration

This topic provides links to collection of tools and controls that help you monitor and adjust VMware Tanzu GemFire performance.

- Deactivate TCP SYN Cookies

  This topic explains how to deactivate TCP SYN cookies. This is required when using Linux systems.

- Improving Performance on vSphere

  This topic provides guidelines for tuning vSphere virtualized environments that host VMware Tanzu GemFire deployments.

- Performance Controls

  This topic provides tuning suggestions of particular interest to developers, primarily programming techniques and cache configuration.

- System Member Performance

  You can modify some configuration parameters to improve system member performance.

- Slow Receivers with TCP/IP

  You have several options for preventing situations that can cause slow receivers of data distributions. 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.

- 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.

- Socket Communication

  Tanzu GemFire processes communicate using TCP/IP. In all cases, communication uses sockets that you can tune to optimize performance.

- Maintaining Cache Consistency

  Maintaining data consistency between caches in a distributed Tanzu GemFire system is vital for ensuring its functional integrity and preventing data loss.

Improving Performance on vSphere

This topic explains how to improve VMware Tanzu GemFire performance on vSphere.

Operating System Guidelines

Review the Host Machine Requirements.

NUMA, CPU, and BIOS Settings

This section provides recommended settings for NUMA, CPU, and BIOS hardware and virtual machines.

- Always enable hyper-threading, and do not overcommit CPU.
- For most production VMware Tanzu 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 activated.
  - Processor Settings: C States deactivated.

> 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 deactivate 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.

  > Deactivating 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 deactivate LRO when interrupt coalescing is deactivated.

This type of tuning benefits Tanzu GemFire workloads, but it can negatively impact other non-VMware Tanzu GemFire workloads that are memory throughput-bound, as opposed to latency sensitive as in the

case of Tanzu GemFire workloads. For more information, see Poor TCP performance might occur in Linux virtual machines with LRO enabled.

- 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. For details about selecting the appropriate type of virtual NIC for your virtual machine, see Choosing a network adapter 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 deactivate 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 its use with DRS.

When vMotion migrations occur, there is an expected temporary drop in the performance of both read-operation and write-operation workloads. These workloads resume their normal rate of operation once the vMotion migration of the servers is completed.

VMware recommends that all vMotion migration activity of VMware Tanzu GemFire members occurs over 10GbE, during periods of low activity and scheduled maintenance windows. Test vMotion migrations in your own environment to assess differences in workload, networking, and scale.

If you want to prevent automatic VMware vSphere vMotion® operations that can affect response times, place VMware vSphere Distributed Resource Scheduler™ (DRS) in manual mode when you first commission the data management system.

VMware vSphere Snapshots

VMware recommend that you disable VMware vSphere Snapshots for clusters running Tanzu GemFire. vSphere Snapshots freeze the VMs and usually, the Tanzu GemFire members are forced out of the cluster as a result of being unresponsive while the Snapshot is taken. You can't troubleshoot this in Tanzu GemFire as there is no visibility of this in the Tanzu GemFire artifacts.

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 VMware Tanzu GemFire servers, also increase the number of virtual machines to maintain a 1:1:1 ratio among the VMware Tanzu GemFire server, the JVM, and the virtual machines.
- Size for a minimum of four vCPU virtual machines with one VMware Tanzu GemFire server running in one JVM instance. This allows ample CPU cycles for the garbage collector, and the rest for user transactions.

- Because VMware Tanzu 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 Tanzu GemFire hosts.
- When sizing memory for a Tanzu 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 VMware Tanzu GemFire

On VMware Tanzu GemFire virtual machines, deactivate vSphere High Availability (HA).

If you are using a dedicated VMware Tanzu GemFire DRS cluster, then you can deactivate HA across the cluster. However, if you are using a shared cluster, exclude Tanzu GemFire virtual machines from vSphere HA.

Additionally, to support high availability, you can also set up anti-affinity rules between the VMware Tanzu GemFire virtual machines to prevent two VMware Tanzu 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 VMware Tanzu GemFire workloads.
- Align disk partitions at the VMFS and guest operating system levels.
- Provision VMDK files as eagerzeroedthick to avoid lazy zeroing for VMware Tanzu GemFire members.
- Use separate VMDKs for VMware Tanzu 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. For more information, see Tuning options for disk I/O performance in Linux 2.6 kernel-based virtual machines.

For more information, see the Performance Tuning for Latency-Sensitive Workloads on VMware vSphere 8 White Paper.

Performance Controls

This topic provides VMware Tanzu GemFire programming techniques, cache configuration suggestions, and other tuning suggestions for developers.

Before you begin, review Tanzu GemFire Basic Configuration and Programming.

- Data Serialization

  In addition to standard Java serialization, Tanzu GemFire offers serialization options that give you higher performance and greater flexibility for data storage, transfers, and language types.

- Setting Cache Timeouts

  Cache timeout properties can modified through the gfsh alter runtime command (or declared in the cache.xml file) and can also be set through methods of the interface, org.apache.geode.cache.Cache.

- Controlling Socket Use

  For peer-to-peer communication, you can manage socket use at the system member level and at the thread level.

- Management of Slow Receivers

  You have several options for handling slow members that receive data distribution. 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.

- 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.

- File System

Data Serialization

In addition to standard Java serialization, VMware Tanzu GemFire serialization options that give you higher performance and greater flexibility for data storage, transfers, and language types.

For more information about data serialization, see Data Serialization.

Setting Cache Timeouts

This topic discuses setting cache timeouts in VMware Tanzu GemFire.

Cache timeout properties can modified through the gfsh alter runtime command (or declared in the cache.xml file) and can also be set through methods of the interface, org.apache.geode.cache.Cache.

To modify cache timeout properties, you can issue the following gfsh alter runtime command. For example:

```
gfsh>alter runtime --search-timeout=150
```

The --search-timeout parameter 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 configurations 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 org.apache.geode.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 settings:

```
gfsh>alter runtime --lock-timeout=30 --lock-lease=60
```

- --lock-timeout. Timeout for object lock requests, specified in seconds. The setting affects automatic locking only, and does not apply to manual locking. The default is one 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, specified in seconds. The setting affects both automatic locking and manual locking. The default is two 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

This topic how to manage socket usage in VMware Tanzu GemFire.

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 use a limited number of shared sockets with other threads or have exclusive access to sockets from a common pool for 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.

By default, conserve-sockets is set to false: each application thread uses a dedicated socket to send to each of its peers and a dedicated socket to receive from each peer. However the sockets are pooled, as threads complete their use of the socket, they are returned back to the pool. If a thread requires a socket and the pool does not have an available socket, a new socket will be created. This default setting optimizes system performance, but does incur significant overhead in terms of sockets, socket buffers, and related system resources.

Keeping conserve-sockets set to false:

- alleviates socket contention between threads
- optimizes distributed ACK operations
- for distributed regions, optimizes the put operation
- for distributed regions, optimizes destroy and invalidate operations for regions and entries
- for partitioned regions, enhances general throughput

Setting conserve-sockets to true:

- reduces demands on system resources
- slows system performance
- increases risk of distributed deadlocks

> 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 org.apache.geode.distributed.DistributedSystem:

- setThreadsSocketPolicy. Sets the calling thread's individual socket policy, overriding the policy set for the application as a whole. If false, the calling thread has exclusive access to the sockets it is using. If true, the calling thread will use sockets concurrently with other threads.
- releaseThreadsSockets. Frees any sockets held by the calling thread. Threads only 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. The sockets when this method is called, will not be returned to the socket pool and instead are closed.

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.

> Calling setThreadsSocketPolicy(false) is meaningful only 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

This topic explains how to manage slow receivers in VMware Tanzu GemFire.

You have several options for handling slow members that receive data distribution. 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.

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.

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.

> 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 cluster.

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 cluster. During the retry cycle, 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.

Entry updates with conflation

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.

> This method of conflation 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 deactivate 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

This topic discusses reasons to increase the ratio of cache hits in VMware Tanzu GemFire.

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 for more information.

File System

This topic describes recommended file systems.

Use XFS or ext4 filesystems when operating on Linux or Solaris platforms. The ext4 filesystem supports preallocation, which benefits disk startup performance. Ext4 generally provides better performance than ext3 while XFS offers comparable performance to ext4.

System Member Performance

This topic links to documentation about VMware Tanzu GemFire system member performance.

You can modify some configuration parameters to improve system member performance.

Before doing so, you should understand Basic Configuration and Programming.

- Member Properties

  Several performance-related properties apply to a cache server or application that connects to the cluster.

- 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.

- Garbage Collection and System Performance

  If your application exhibits unacceptably high latencies, you might improve performance by modifying your JVM's garbage collection behavior.

Member Properties

This topic discusses the impact of changing several performance-related properties in VMware Tanzu GemFire.

These performance-related properties apply to a cache server or application that connects to the cluster.

- statistic-sampling-enabled. This property is deprecated as it will be enabled by default. It will be removed in upcoming releases. 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.

JVM Memory Settings and System Performance

This topic discusses JVM memory settings and system performance with VMware Tanzu GemFire.

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 Tanzu 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
```

- Off-heap memory size—For applications that use off-heap memory, specifies how much off-heap memory to allocate. Setting off-heap-memory-size is prerequisite to enabling the off-heap capability for individual regions. For example:

  ```
  gfsh>start server --name=server-name --off-heap-memory-size=200G
  ```

  See Using Off-heap Memory for additional considerations regarding this parameter.

- Lock memory—On Linux systems, you can prevent heap and off-heap memory from being paged out by setting the lock-memory parameter to true. For example:

  ```
  gfsh>start server --name=server-name --off-heap-memory-size=200G --lock-memory=true
  ```

  See Locking Memory for additional considerations regarding this parameter.

Garbage Collection and System Performance

This topic discusses garbage collection and system performance, and explains how to modify your JVM's garbage collection behavior to improve VMware Tanzu GemFire performance.

If your application exhibits unacceptably high latencies, 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. You can reduce the impact of garbage collection in two ways:

- Optimize garbage collection in the JVM heap.
- Reduce the amount of data exposed to garbage collection by storing values in off-heap memory.

> 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.
>
> 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.

Optimizing Garbage Collection

To reduce latency and improve responsiveness, you can tune your JVM's garbage collection behavior using several startup options. The following flags are commonly used to improve garbage collection performance.

- Use concurrent and parallel garbage collection to reduce pause times and improve throughput:

```
-XX:+UseConcMarkSweepGC -XX:+UseParNewGC
```

- Improve response time consistency by pre-touching the heap at JVM startup. This forces the JVM to allocate and initialize all heap memory upfront, avoiding incremental page faults during execution. While this may increase startup time, it reduces runtime latency and improves garbage collection performance.

```
-XX:+AlwaysPreTouch
```

- If your application uses Remote Method Invocation (RMI), the JVM triggers explicit garbage collection every 60 seconds. This can introduce unnecessary latency. You can disable this behavior with:

```
-XX:+DisableExplicitGC
```

> Test carefully to ensure your JVM can handle GC load without explicit triggers.

Using Off-heap Memory

You can improve the performance of some applications by storing data values in off-heap memory. Certain objects, such as keys, must remain in the JVM heap. See Managing Off-Heap Memory for more information.

GemFire Health Checklist

This section describes six key health factors in Tanzu GemFire that help assess system stability. Each focuses on a critical resource such as Java heap memory, CPU, disk space, or cluster communication and includes potential impacts, monitoring tools, and recommended remedies. GemFire provides several tools for observing these factors, including system logs, statistics, the GemFire Shell (gfsh), and the GemFire Management Console. Together, these resources provide a structured way to detect, understand, and resolve common system issues.

Health Factor Name	Description
JVM Pauses High	A GemFire task may appear to pause for a period of time. If the pause lasts several seconds or more, it can lead to systemic issues
CPU Steal High	When a VM (guest) doesn't receive its allotted CPU resources, system performance can degrade and cluster members may be at risk of being lost.
Resource Manager Health Factors	When a Java program runs out of heap memory, it crashes. The GemFire Resource Manager helps prevent this by managing memory use.
Reply Timeouts High	When responses to GemFire peer-to-peer requests are delayed, system performance and even system correctness may be compromised.
Partitioned Region Redundancy Low	If a partitioned region drops below its redundancy target for a long time, applications relying on that region's data are vulnerable to loss of service.
Disk Low	When a volume storing GemFire disk stores fills up, applications are vulnerable to loss of service.

JVM Pauses High

This topic explains how long JVM pauses can cause problems in Tanzu GemFire. It shows how to spot the issue using system stats and what might be causing it, like CPU load or garbage collection.

A GemFire server runs many concurrent tasks on the Java Virtual Machine (JVM). Usually each task makes steady progress over time, as if it were executing on a dedicated machine. However, certain issues can degrade this progress and cause tasks to stop working temporarily, this is commonly referred to as a 'JVM pause'. If these pauses last several seconds or more, they can cause systemic performance problems.

Detecting JVM Pauses

One way to detect a JVM pause is by monitoring the StatSampler.jvmPauses statistic. An increase in this value indicates that the GemFire statistics-gathering task is running behind schedule. From the perspective of that periodic task, the JVM was 'paused'. When one task is paused, it's very often the case that many others are too.

Rising StatSampler.jvmPauses values should be treated as an important signal of potential performance issues within the JVM or system environment.

Impact

GemFire cluster members (cache servers and locators) monitor each other to maintain cluster health. If a member stops receiving communications from the member it's monitoring, an escalation process is initiated. This process can result in the affected member being excluded from the cluster and shut down.

Membership Failure Detector
This escalation process is managed by the *membership failure detector*, the GemFire subsystem responsible for making exclusion decisions. It detects isolated problems early, so that the health of the overall cluster is preserved.

Whether the membership failure detector improves the situation by excluding a member depends on the root cause of the communication failure:

- Improve: If the member is running on a host with a failing network interface, excluding the member and starting a replacement on a different host can improve overall cluster health.
- Worsen: If the root cause is high CPU load due to legitimate GemFire application demand, then excluding that member increases the load on the remaining cluster members. In this case, the failure detector may have made the problem worse.

JVM Pauses and Communication Failures
High JVM pauses are often accompanied by network communication issues. When a JVM experiences prolonged pauses, its network tasks may fail to send or receive data in time. If the membership failure detector notices a gap in communication, it will remove the member from the cluster. This appears in logs as a ForcedDisconnectException.

Behavior After Exclusion
What happens after a member is excluded depends on whether auto-reconnect is enabled (which it is by default):

- Auto-reconnect enabled: The process will attempt to rejoin the cluster automatically.
- Auto-reconnect disabled:
    - If you're using an external system like Elastic Application Runtime or Tanzu GemFire on Kubernetes, a new member may be started to replace the failed one.

- If no external process restarts the failed member, the cluster continues running with one member missing. This reduces both performance and data redundancy, and might lead to data loss.

Where to Look

To monitor for conditions that might lead to member exclusion, such as prolonged JVM pauses, GemFire provides several diagnostic statistics.

Statistics-gathering Task
The statistics-gathering task in GemFire runs periodically, by default once per second. This interval can be adjusted using the statistic-sample-rate property. During each run, the task collects system metrics before sleeping until the next sampling period. If the statistics-gathering task falls behind schedule by three or more sampling intervals, the jvmPauses statistic is incremented to signal potential JVM pauses affecting performance.

Key Supporting Metrics

To catch early signs of this degradation, monitor two key metrics within the StatSampler statistic type: sampleTime and delayDuration. These metrics help determine whether the statistics-gathering task is running on time or falling behind.

- delayDuration is a gauge that measures how long the sampler sleeps between collections. In a healthy system, this value equals statistic-sample-rate minus the most recent change in sampleTime. For example, if sampleTime increases by approximately 20 milliseconds per second, and statistic-sample-rate is set to the default 1000 milliseconds, then delayDuration should be approximately 980 milliseconds. If delayDuration reaches three times the statistic-sample-rate, GemFire will increment jvmPauses.
- sampleTime is a counter that accumulates time spent taking samples. This typically is a small fraction of the sampling period, for example, 20 milliseconds per second in a healthy system. If sampleTime approaches the statistic-sample-rate, it indicates a problem. When it reaches three times the statistic-sample-rate, GemFire will increment jvmPauses.

Analyzing sampleTime Variation

In addition to monitoring increases in sampleTime, it's often useful to examine variation in this statistic. High variability can indicate underlying performance issues, even if average values seem acceptable.

One useful measure of variability is the ratio of the standard deviation to the average, this is referred to as the *coefficient of variation*. A higher value suggests greater instability in system behavior.

TIP: In VSD, to view the standard deviation and average for a charted line, select the line and choose Line > Log Info

Example: Comparing Healthy and Unhealthy Servers
The charts below show sampleTime values per second from two GemFire servers:

- The first chart shows a healthy GemFire server. The coefficient of variation is approximately 0.3, calculated as:
  - Average = 3.6
  - Standard Deviation = 1.0
  - Coefficient of Variation = 1.0 / 3.6 = 0.3
- The second chart shows an unhealthy GemFire server experiencing high CPU steal time. The coefficient of variation is approximately 1.8, calculated as:

- Average = 5.8
- Standard Deviation = 10.6
- Coefficient of Variation = 10.6 / 5.8 = 1.8

Viewing Standard Deviation and Other JVM Metrics

VSD: Examine the following statistics:

- StatSampler.jvmPauses
- StatSampler.sampleTime
- StatSampler.delayDuration

gfsh: Examine the Alerts section in the output of the describe region --name=NAME --verbose command. For more information, see describe in the gfsh Command Help.

System logs: Look for messages like:

```
Statistics sampling thread detected a wakeup delay of 14318 ms, indicating a possible resource issue. Check the GC, memory, a
nd CPU statistics.
```

```
Failure detection heartbeat-generation thread overslept by more than a full period. Asleep time: 1,000,000,001 nanoseconds. Per
iod: 500,000,000 nanoseconds.
```

Remedies

High JVM pauses are typically caused by one or both of the following:

- Hypervisor virtualization pauses. For more information, see CPU Steal High.
- Java garbage collection (GC) pauses

If CPU Steal has been ruled out, the most likely cause is Java GC pause activity.

Understanding Garbage Collection Pauses
The Java platform offers many different garbage collectors, each providing different performance and scalability tradeoffs. A key metric is GC latency, this is the amount of time a Java application is paused during GC. Modern collectors such as Shenandoah and ZGC deliver pause times in the milliseconds or sub-millisecond range. G1GC can also be configured to achieve strong performance in GemFire applications.

GC Metrics in GemFire
GemFire captures basic garbage collector metric statistics with GarbageCollectorMXBean. For each garbage collector phase, GemFire provides a corresponding VMGCStats instance:

Collector	VMGCStats Instance Name
G1GC	G1 Young Generation
G1GC	G1 Old Generation
ZGC	ZGCCycles
ZGC	ZGCPauses

Each VMGCStats reports:

- VMGCStats.collections: Number of GC events recorded
- VMGCStats.collectionTime: Total time spent in GC

In older collectors like the now deprecated CMS, VMGCStats.collectionTime was a good indicator of application thread pause time because GC paused all application threads. However, in modern collectors such as G1GC and ZGC, much of the work is performed concurrently. As a result, VMGCStats.collectionTime no longer directly indicates application pause duration.

However, collectionTime can still serve as a useful signal. If it increases sharply during specific operations, or seems high relative to available CPU resources, further investigation into garbage collector behavior

might be warranted.

Use Resource Manager
The GemFire documentation on the Resource Manager has detailed advice on configuring various garbage collectors. For information about the Resource Manager, configuring garbage collectors, heap size, and tuning settings, see Using the GemFire Resource Manager in the Managing Heap Memory topic.

Use Yourkit
A Java profiler can help you find code that generates lots of garbage. YourKit has particularly useful GC profiling capability.

Configure your Garbage Collector
If you are using the GemFire Resource Manager to evict infrequently-used region data, follow the advice in the Managing Heap Memory topic to understand how to configure your particular garbage collector.

CPU Steal High

This topic covers how high CPU steal time can impact GemFire performance and offers guidance on monitoring and reducing it through system stats and Linux configuration tips.

GemFire is often deployed in a virtual machine (VM), such as the one provided by the VMware ESXi hypervisor. A hypervisor can run multiple VMs on a single host machine. However, if too many VMs share the same host, CPU resources can become over committed, causing performance degradation in the VMs.

If the VM guest operating system is Linux-based, the LinuxSystemStats.cpuSteal statistic indicates the percentage of CPU time 'stolen' from the VM. A non zero cpuSteal value, especially when StatSampler.jvmPauses is also increasing, indicates a problem that requires attention.

For more information, see Operating System Statistics - Linux in the Statistics List topic.

Impact

High CPU steal impacts GemFire in two ways:

- Slower performance: The effective CPU power available to the application is reduced, resulting in slower GemFire performance.
- Interruptions in processing: GemFire processes rely on uninterrupted access to CPU resources. When CPU steal is high, there can be interruptions caused by periods when little or no CPU is available. During an interruption, GemFire increments the StatSampler.jvmPauses counter. Interruptions can lead GemFire's internal health monitoring system to conclude that the process or the machine it is operating on is experiencing a fault. As a result, the process might be excluded from the cluster and shut down. What happens after a member is excluded depends on whether auto-reconnect is enabled (which it is by default):
    - Auto-reconnect enabled: The process will attempt to rejoin the cluster automatically.
    - Auto-reconnect disabled:
        - If you're using an external system like Elastic Application Runtime or Tanzu GemFire on Kubernetes, a new member may be started to replace the failed one.
        - If no external process restarts the failed member, the cluster continues running with one member missing. This reduces both performance and data redundancy, and might lead to data loss.

Where to Look

- GemFire Management Console: Examine % Steal Time in the Cluster graph. For more information, see Monitoring Clusters in the VMware Tanzu GemFire Management Console documentation.
- Visual Statistics Display (VSD): View the LinuxSystemStats.cpuSteal statistic. The following section examines CPU steal time and whether your system supports it.

CPU steal time

LinuxSystemStats.cpuSteal is the percentage of CPU processing time 'stolen' from the guest operating system by the hypervisor. GemFire derives this metric from the steal_time counter in /proc/stat. For more information, see /proc/stat explained in the LinuxHowtos documentation.

> This feature must be supported by both the kernel and the guest operating system.

Not all Linux-based operating systems can track CPU steal time. If the operating system is capable of tracking CPU steal time, hypervisor-specific configuration may be required. Check the documentation for your hypervisor and Linux variant for configuration instructions.

> Support for CPU steal time accounting was added to Linux kernel version 5.7 in 2020.

VMware ESXi

- PhotonOS guests have CPU steal time tracking enabled by default.
- Other guest operating systems require the VM configuration option: stealclock.enable = "TRUE".

Examples of guest operating systems that can track CPU steal time under VMware ESXi (with proper configuration) include:

- SUSE Linux Enterprise Server version 15
- SUSE Linux Enterprise 12 SP5

> If a VM does not support tracking CPU steal time, the steal_time counter in /proc/stat will always be 0 and as a result, GemFire will report 0% stealTime. This does not mean your system is free of CPU steal time. Check the documentation for your hypervisor and Linux-based guest operating system to find out how to enable CPU steal time accounting.

Remedies for High CPU Steal

High CPU steal typically indicates overprovisioning, where too many virtual CPU cores are allocated relative to the available physical cores. This overprovisioning may only become apparent under increased system demand.

The solutions are:

- Reduce vCPU count: Limit the number of virtual cores competing for physical cores on the machine.
- Reallocate non-critical workloads:
    - Move non-GemFire VMs to other physical machines.

    - Reduce CPU allocation for less critical VMs.
- Upgrade hardware: In some cases, it may be necessary to move the GemFire VMs to more capable machines.

Resource Manager Health Factors

This topic explains how the Resource Manager responds to rising memory pressure, including eviction behavior, critical thresholds, and the potential impact on application operations.

When a Java program runs out of heap memory, it crashes. To prevent this, the GemFire Resource Manager monitors heap usage and takes proactive steps. If heap utilization exceeds a configured threshold (the eviction threshold), the Resource Manager starts a background task that evicts infrequently-used objects from the cache to free memory. Eviction continues until utilization returns to a healthy level. If heap usage climbs further and exceeds the critical threshold, GemFire halts cache additions to reduce memory pressure, allowing eviction and garbage collection to restore stability.

The following diagram depicts the various states from the perspective of the GemFire Resource Manager:

Impact

When a GemFire server crosses the eviction threshold, it begins evicting infrequently-used cache entries in the background. In this state, any data added to the cache triggers proportional evictions to maintain memory balance.

If the heap enters the critical state, operations that increase memory usage, such as PUT are rejected. Clients attempting these operations will receive a LowMemoryException. Meanwhile, the server continues background eviction in an attempt to recover.

Operating in the critical state risks heap exhaustion. If the heap is fully consumed, the server may crash. Depending on your redundancy configuration, this can result in data loss.

Where to Look

gfsh: Examine the Alerts section in the output of the describe region --name=NAME --verbose command to determine if your system is currently in either the eviction state or the critical state. For more information, see describe in the gfsh Command Help.

Visual Statistics Display (VSD): To assess the state of your Java heap utilization in the past, whether it was healthy (below the eviction threshold), in eviction, or in a critical state, review the following

ResourceManagerStats counters:

- evictionStartEvents
- evictionStopEvents
- heapCriticalEvents
- heapSafeEvents

For more information, see Statistics List in the Statistics List topic.

System logs: Look for messages containing:

- above heap eviction threshold
- below heap eviction threshold
- above heap critical threshold
- below heap critical threshold

Determining Historical Heap States from ResourceManager Statistics

You can retrospectively determine the state of a GemFire server's Java heap utilization by analyzing specific counters in the ResourceManagerStats. The system may have been in one of three possible states:

- Healthy: Heap usage below the eviction threshold.
- Eviction: Heap usage above the eviction threshold; evictions in progress.
- Critical: Heap usage above the critical threshold; evictions in progress and data additions were rejected

The following pseudo-code outlines the logic

```
if (heapCriticalEvents - heapSafeEvents) > 0 then
   state = "Critical"
else if (evictionStartEvents - evictionStopEvents) > 0 then
   state = "Eviction"
else
   state = "Healthy"
```

- Critical State: If the number of heapCriticalEvents exceeds the number of heapSafeEvents, the system was in the critical state at some point. In this state, both entry evictions and data admission controls are triggered.
- Eviction State: If the system was not in the critical state but the number of evictionStartEvents exceeds the number of evictionStopEvents, the system was in the eviction state. This indicates that heap usage exceeded the eviction threshold, and evictions were active.
- Healthy State: If neither condition is met, the system was in the healthy state, with heap usage below the eviction threshold.

In addition to the statistics, the Resource Manager logs transition into and out of these states in the GemFire system log. Look for the following log messages:

- above heap eviction threshold
- below heap eviction threshold
- above heap critical threshold
- below heap critical threshold

These logs provide additional confirmation of the system's state transitions over time.

Remedies

Follow the advice in Managing Heap Memory to configure GemFire and to configure your particular garbage collector.

Reply Timeouts High

This topic explains how delayed peer-to-peer responses in GemFire can trigger reply timeout warnings and impact system behavior. It shows how to identify and troubleshoot these timeouts using statistics, log messages, and configuration settings.

When responses to GemFire peer-to-peer (P2P) requests are delayed, system performance and even system correctness might be compromised.

An increase in the DistributionStats.replyTimeouts counter indicates the 'reply timeouts high' condition. This does not mean that a request has been abandoned, or that a subsequent reply will be ignored. Instead, it means that the alerting threshold has been exceeded. The requesting task will continue to wait for a response, and when one arrives, it will be processed. For more information, see Distributed System Messaging in the Statistics List topic.

The alerting threshold is determined by the ack-wait-threshold property. For more information, see GemFire Properties.

Impact

P2P messaging supports communication between cluster members, including cache servers and locators. One-way messages are sufficient for notifications that do not require a response. However, many of the GemFire core functionalities depend on request-reply messaging, where the receiving member must return a response. This includes:

- Cluster startup and shutdown
- Distributed locking
- Region management
- Cache operations
- Transactions
- WAN replication

Where to Look

GemFire Management Console: Monitor the replyWaitsInProgress statistic.

GemFire Visual Statistics Display (VSD): View the DistributionStats.replyTimeouts statistic. A recent increase in this statistic indicates there is a problem.

GemFire system logs: System logs can help identify the cause of an increase in the replyTimeouts statistic. For more information, see Log Messages and Solutions.

Tracking Reply Timeouts and Log Messages

- When a reply is delayed, the replyTimeouts counter is incremented. A log message similar to one of the following is generated indicating that a request for a distributed lock has exceeded the ack-wait-threshold:

```
15 seconds have elapsed while waiting for a direct reply response from host2(member-alpha:104149)<ec><v3>:41000
```

```
15 seconds have elapsed while waiting for replies
```

```
15 seconds have elapsed while waiting for replies: <DLockRequestProcessor 33636 waiting
```

Where:

- The logging member is waiting on a response from host2(member-alpha:104149)<ec><v3>:41000.
- 15 seconds is the value of the ack-wait-threshold configuration parameter. If you've changed that parameter you'll see a different time here.
- DLockRequestProcessor 33636 indicates the subsystem affected by the delayed response.

- When a delayed reply is received, a message similar to one of the following is logged:

```
Finished waiting for direct reply from host2(member-alpha:104149)<ec><v3>:41000
```

```
DLockRequestProcessor wait for replies completed
```

Remedies

A 'reply timeouts high' condition can be caused by application code issues, such as long-running functions or queries, or by GemFire system configuration.

Functions: GemFire maintains an instance of the FunctionServiceStatistics statistic type, for each GemFire function that your application deploys. This statistic type provides counters that assess function performance. For more information, see GemFire Properties and Statistics List.

Queries: If you suspect a query is slow:

- Use the gfsh list queries command to identify the currently executing queries. The elapsed time for each query is shown.
- For infrequent or fast queries, add the <trace> keyword to generate a log message at query start and completion. For more information, see Query Debugging.
- For slow queries, see Performance Considerations for advice on tuning them.

Log entries: If reply timeouts persist after addressing slow functions and queries, search for log entries containing have elapsed while waiting for replies.

For example,

```
[warn 2024/03/02 15:23:18.961 PST bridgegemfire1_host1_67433 <vm_1_thr_1_bridge1_host1_67433> tid=0x19] 15 seconds h
ave elapsed while waiting for replies: <CreateRegionProcessor$CreateRegionReplyProcessor 4 waiting for 1 replies from [host2
(member-alpha:104149)<ec><v3>:41000]> on host1(member-beta:67433)<ec><v2>:41001 whose current membership list is: [<
membership-list>]
```

CreateRegionProcessor indicates that the request involves region creation.

This log entry indicates that member host2 failed to respond promptly to the request from host1. If multiple logs point to delays from the same member, investigate that member's configuration or system load.

Quiet the warnings: If your system is performing satisfactorily despite reply timeouts, consider increasing the ack-wait-threshold to reduce or suppress warning messages.

Lower GemFire Versions

Logging behavior changed in GemFire 10.2.0. If you are running a version of GemFire below 10.2.0, here's what to look for.

For certain message types, the log entry looks like this:

```
15 seconds have elapsed while waiting for a response from host2(member-alpha:104149)<ec><v3>:41000
```

Where

- 15 seconds is the value of the ack-wait-threshold configuration parameter. If you've changed that parameter you'll see a different time here.
- The logging member is waiting on a response from host2(member-alpha:104149)<ec><v3>:41000

When a reply is received, after a reply timeout, a message like this is logged:

```
Finished waiting for reply from host2(member-alpha:104149)<ec><v3>:41000
```

For other message types, a late reply results in a log message like this:

```
15 seconds have elapsed while waiting for replies
```

These log messages often provide additional information about which subsystem is affected by the delayed response. For example, this message indicates that a request for a distributed lock has exceeded the ack-wait-threshold:

```
15 seconds have elapsed while waiting for replies: <DLockRequestProcessor 33636 waiting
```

When a reply is received, a message like this is logged:

```
DLockRequestProcessor wait for replies completed
```

Where DLockRequestProcessor is the reply processor type. This can vary and often provides useful insight into which GemFire subsystem is affected.

More information

Log Messages and Solutions

GemFire Properties - gemfire.properties and gfsecurity.properties

status command

Statistics List

Query Debugging

The Most Useful Statistics for Troubleshooting VMware GemFire Deployments blog.

Partitioned Region Redundancy Low

This topic explains how low redundancy in a partitioned region can lead to data loss or service issues in Tanzu GemFire. It covers how to detect the problem and restore redundancy by adding data stores.

When a partitioned region's redundancy falls below the configured target for an extended period, there is a risk of data loss. Applications relying on the region's data may experience service disruptions. The

condition of ‘partitioned region redundancy low’ is indicated when the PartitionedRegionStats.<region-name>.lowRedundancyBucketCount value is greater than zero.

Impact

A GemFire cache server that stores data for a region is called a data store for that region. In contrast, an accessor server can access region data but does not store it directly.

Data in a partitioned region remains available to applications even if one or more data stores are lost. As long as one copy of the data survives, the data will be available to applications. By configuring redundant-copies in the partition-attributes for the region and ensuring a sufficient number of data stores, any level of availability can be achieved.

Redundancy Conditions

When bucket redundancy drops below the configured level, GemFire will set the PartitionedRegionStats.<region-name>.lowRedundancyBucketCount to the number of buckets that are currently below their redundancy target.

There are two scenarios where PartitionedRegionStats.<region-name>.lowRedundancyBucketCount will be greater than zero:

- When a new partitioned region is created with redundancy on a GemFire cluster that doesn’t have enough data stores to provide that redundancy.
- When a GemFire cluster that formerly had enough data stores to satisfy the redundancy requirement of the region, loses one or more members.

In both cases, once redundancy falls below the configured level, applications relying on data from that region are at risk of disruption or data loss.

> For convenience, a statistic: PartitionedRegionStats.<region-name>.configuredRedundantCopies shows the value of the redundant-copies configuration for the region.

For more information, see cache Element Reference.

Where to Look

- gfsh:
  - status redundancy command
  - Look for a redundancy value of ‘Not Satisfied’ in the output of the describe region command.
- GemFire Management Console:
  - Check for a redundancy entry in the Regions table on the Regions tab. For more information, see Regions in the VMware Tanzu GemFire Management Console documentation.
- GemFire Visual Statistics Display: Check the following statistics:
  - PartitionedRegionStats.<region-name>.lowRedundancyBucketCount
  - PartitionedRegionStats.<region-name>.configuredRedundantCopies

- System logs: Look for log messages like Redundancy has dropped below <n> configured copies. For more information, see Log Messages and Solutions.

For more information, see Partitioned Region Statistics on Redundancy in the Statistics List topic.

Remedies

If one or more cache servers acting as data stores for a partitioned region leave the cluster, due to a failure or shutdown, lowRedundancyBucketCount might increase. Replace the lost data stores until the total number of data stores meets or exceeds the redundant-copies setting for the region.

Disk Low

This topic explains how a full disk store in Tanzu GemFire can disrupt server operations and how to prevent it through monitoring and configuration.

When a disk store for Tanzu GemFire becomes full, it can cause application disruptions.

When the DiskDirStatistics.volumeFreeSpace statistic associated with a disk store approaches zero, it indicates the 'disk low' condition.

For more information, see Disk Space Usage (DiskDirStatistics) in the Statistics List topic.

Impact

A Tanzu GemFire disk store manages a persistent region's disk storage usage. It defines one or more directories where region data is stored, with each directory located on a disk or storage volume. If a cache server needs to write region data to disk but the associated volume is full, the server will shut down.

For more information, see Disk Store Configuration Parameters and Preventing and Recovering from Disk Full Errors.

Where to Look

Use Visual Statistics Display(VSD) or System logs to determine if your system is approaching or has reached the 'disk low' condition.

VSD: For each directory in a disk store configuration, there is an instance of the DiskDirStatistics.volumeFreeSpace statistic in VSD. When the value for this statistic reaches zero, this indicates a 'disk low' condition. For more information, see Disk Space Usage (DiskDirStatistics) in the Statistics List topic.

System logs: Tanzu GemFire provides monitoring and alerting of the 'disk low' condition. It is configured at the disk store level. This provides an early warning before the storage volume runs out of free space.

When configuring a disk store you can specify the --disk-usage-warning-percentage and --disk-usage-critical-percentage parameters. When these thresholds are reached, messages are logged in the system log. For example,

```
warn…The disk volume volume-name for disk store store-name has exceeded the warning usage threshold and is N% full
error…The disk volume volume-name for disk store store-name has exceeded the critical usage threshold and is N% full
```

For example:

```
[warn 2014/06/06 06:54:02.486 PDT  <Disk Store Monitor-diskStore1> tid=0x16] The disk volume /volume1 for disk store diskSto
re1 has exceeded the warning usage threshold and is 97% full

[error 2014/06/06 06:54:02.486 PDT  <Disk Store Monitor-diskStore1> tid=0x16]  The disk volume C:\Users\systemX for disk stor
e local_disk_store has exceeded the critical usage threshold and is 100% full
```

For more information, see Configuring Disk Free Space Monitoring, and Disk Store Configuration Parameters.

Remedies

When you notice that disk is low you have two options.

- You can increase the storage capacity of the storage volume. Depending on your operating system and storage volume type, it may be possible to do this without an interruption of service.
- If you can't increase the storage capacity of the storage volume, the alternative is to make one or more new storage volumes available and to then add one or more data store servers to your GemFire cluster. After such reconfiguration it is necessary to initiate a cluster rebalance operation. Rebalancing will spread existing persistent region data over more servers, so each server will have less data to manage. For more information, see Running VMware Tanzu GemFire Server Processes.

Slow Receivers with TCP/IP

This topic describes the options available for preventing situations that can cause slow receivers of data distributions when using VMware Tanzu GemFire. 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.

Before you begin, review Tanzu GemFire Basic Configuration and Programming.

- Preventing Slow Receivers

  During system integration, you can identify and eliminate potential causes of slow receivers in peer-to-peer communication.

- 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 cluster.

Preventing Slow Receivers

This topic explains how to prevent slow receivers in VMware Tanzu GemFire.

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 Tanzu GemFire members have enough CPU available to them. Do not run any other heavyweight processes on the same machine.

The machines that host Tanzu 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 Tanzu GemFire system or by contention between the Tanzu GemFire traffic and other traffic on your network. Consider whether more subnets are needed to separate the Tanzu GemFire administrative traffic from Tanzu GemFire data transport and to separate all the Tanzu GemFire traffic from the rest of your network load.

Plan for Growth

Upgrade the infrastructure to the level required for acceptable performance. Analyze the expected Tanzu 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 Tanzu GemFire application and cache server processes can handle the expected load.

Managing Slow Receivers

This topic explains how to manage slow receivers in VMware Tanzu GemFire.

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 cluster.

During the retry cycle, Tanzu 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 cluster.

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.

> These slow receiver options are deactivated in systems using SSL. See SSL.

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 cluster.
- 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 cluster.

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 cluster. 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.

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 Tanzu GemFire failure detection code will eventually cause the member to leave the cluster 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 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 cluster.
- 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 is not 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 is not responding, a severe warning alert is issued, indicating that the member may be disconnected from the cluster. 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. Tanzu 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.

Events described below

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 cluster by other members. Typically, this happens when a member becomes unresponsive and does not respond to heartbeat requests within the member-timeout period, or when ack-severe-alert-threshold has expired without a response from the member.

> This is marked as a region operation.

Other members see the normal membership notifications for the departing member. For instance, RegionMembershipListeners receive the afterRemoteRegionCrashed notification, and SystemMembershipListeners receive the memberCrashed notification.

Slow distributed-ack Messages

This topic describes distributed-ack messages in VMware Tanzu GemFire.

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 use TCP, check whether you have socket conservation enabled for your members. It is configured by setting the Tanzu 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 deactivate 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.
    - Reduce the socket-buffer-size in gemfire.properties.

Socket Communication

This topic describes how VMware Tanzu GemFire processes communicate using TCP/IP and provides links to documentation that provide details. All process communications use sockets that you can tune to optimize performance.

The adjustments that you make to tune your Tanzu GemFire communication may reach operating system limits. If this happens, consult 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 Tanzu GemFire Basic Configuration and Programming.

- Setting Socket Buffer Sizes

  When you determine buffer size settings, you try to strike a balance between communication needs and other processing.

- Ephemeral TCP Port Limits

  By default, Windows' ephemeral ports are within the range 1024-4999, inclusive.You can increase the range.

- Making Sure You Have Enough Sockets

  The number of sockets available to your applications is governed by operating system limits.

- TCP/IP KeepAlive Configuration

  Tanzu GemFire supports TCP KeepAlive to prevent socket connections from being timed out.

- 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.

Setting Socket Buffer Sizes

This topic discuses setting socket buffer sizes in VMware Tanzu GemFire.

When you determine buffer size settings, you must 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 do not 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 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
TCP / IP	---	---
Peer-to-peer send/receive	gemfire.properties	socket-buffer-size
Client send/receive	cache.xml <pool>	socket-buffer-size
Server send/receive	gfsh start server or cache.xml <CacheServer>	socket-buffer-size

TCP/IP Buffer Sizes

If possible, your TCP/IP buffer size settings should match across your Tanzu 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 cluster.
- 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>
```

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:

```
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, consult with your system administrator about adjusting the operating system limits.

Ephemeral TCP Port Limits

This topic explains how to increase ephemeral TCP port limits for use with VMware Tanzu GemFire.

By default, Windows' ephemeral ports are within the range 1024-4999, inclusive. You can increase the range.

If you repeatedly receive the following exception:

```
java.net.BindException: Address already in use: connect
```

And your system experiences 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 seen only 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.

Making Sure You Have Enough Sockets

This topic discusses managing the number of sockets you have in VMware Tanzu GemFire.

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 Tanzu GemFire and ways to limit socket consumption in your Tanzu GemFire members.

Socket Sharing

You can configure socket sharing for peer-to-peer:

- 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 property conserve-sockets to true. To achieve maximum throughput, however, we recommend that you use the default value of false.

  At the thread level, developers can override this setting by 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.

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 and instead use dedicated sockets from a pool, you can use the socket-lease-time to make sure that no socket sits idle for too long. When a socket remains in the pool unused for this time period, the system automatically closes that socket. If the pool does not have a socket available when needed, it creates a new socket.
- 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 or a client
- 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 on 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 cluster 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 cluster 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, known as thread-owned sockets, refers to a thread owning a socket while sending a message on it. When the message send completes, the socket is released back to the socket pool. For calculation of thread-owned sockets, an additional socket is 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. The calculated number of sockets is worst case, assuming all threads are using all sockets concurrently.

In this table:

- M is the total number of members in the cluster.
- 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

Peer Member Socket Description	Number Used
Shared sockets (2 in and 2 out) Threads that share sockets use these.	4 * (M-1)
This member's thread-owned sockets (1 for each thread, for each peer member).	(T) * (M-1)
Other members' thread-owned sockets that connect to this member (1 in 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)

> 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 servers 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 cluster.

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 * number of threads in a server that service client pool connections) * (M-1) These threads do not share sockets.

Server Socket Description	Number Used
Subscription connections	2 * number of client subscription connections to this server

With client/server installations, the number of client connections to any single server is undetermined, but Tanzu GemFire's server load balancing and conditioning keeps the connections fairly evenly distributed among servers.

Servers are peers in their own cluster 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 cluster, it has the additional socket requirements as noted in the Peer-to-Peer section of this topic.

TCP/IP KeepAlive Configuration

VMware Tanzu GemFire supports TCP KeepAlive to prevent socket connections from being timed out.

The gemfire.setTcpKeepAlive system property prevents connections that appear idle from being timed out, for example, by a firewall. When configured to true, Tanzu GemFire enables the SO_KEEPALIVE option for individual sockets. This operating system-level setting allows the socket to send verification checks (ACK requests) to remote systems in order to determine whether or not to keep the socket connection alive.

> The time intervals for sending the first ACK KeepAlive request, the subsequent ACK requests and the number of requests to send before closing the socket is configured on the operating system level.

By default, this system property is set to true.

TCP/IP Peer-to-Peer Handshake Timeouts

This topics explains how to alleviate connection handshake timeouts for TCP/IP connections in VMware Tanzu GemFire by increasing the connection handshake timeout interval with the system property p2p.handshakeTimeoutMs.

The default setting is 59000 milliseconds.

The following sets the handshake timeout to 75000 milliseconds for a Java application:

```
-Dp2p.handshakeTimeoutMs=75000
```

You can pass this property to the cache server on the gfsh command line:

```
gfsh>start server --name=server_name --J=-Dp2p.handshakeTimeoutMs=75000
```

Configuring Sockets in Multi-Site (WAN) Deployments

This topic explains how to configure sockets in multi-site (WAN) deployments of VMware Tanzu GemFire.

When you determine buffer size settings, you try to strike a balance between communication needs and other processing.

This table lists the settings for gateway relationships and protocols, and tells where to set them.

Protocol / Area Affected	Configuration Location	Property Name
TCP / IP	---	---
Gateway sender	gfsh create gateway-sender or cache.xml <gateway-sender>	socket-buffer-size
Gateway receiver	gfsh create gateway-receiver or cache.xml <gateway-receiver>	socket-buffer-size

TCP/IP Buffer Sizes

If possible, your TCP/IP buffer size settings should match across your installation. At a minimum, follow the guidelines listed here.

- 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 unsynchronized with the sender and the receiver will be unaware of the condition.

  The gateway sender's socket-buffer-size attribute should match the gateway receiver's socket-buffer-size attribute for all gateway receivers that the sender connects to, as in these example cache.xml snippets:

  ```
  Gateway Sender Socket Buffer Size cache.xml Configuration:

  <gateway-sender id="sender2" parallel="true"
   remote-distributed-system-id="2"
   socket-buffer-size="42000"
   maximum-queue-memory="150"/>

  Gateway Receiver Socket Buffer Size cache.xml Configuration:
  <gateway-receiver start-port="1530" end-port="1551"
   socket-buffer-size="42000"/>
  ```

> WAN deployments increase the messaging demands on a Tanzu GemFire system. To avoid hangs related to WAN messaging, always use the default setting of conserve-sockets=false for Tanzu GemFire members that participate in a WAN deployment.

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 clusters and have the additional socket requirements as noted in the Peer-to-Peer section above.

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> or use the org.apache.geode.cache.client.PoolFactory.setReadTimeout method. For a gateway sender or gateway receiver, see WAN Configuration.

Maintaining Cache Consistency

This topic explains how to maintain cache consistency in VMware Tanzu GemFire.

Maintaining data consistency between caches in a distributed Tanzu GemFire system is vital for ensuring its functional integrity and preventing data loss.

General Guidelines

Before Restarting a Region with a Disk Store, Consider the State of the Entire Region

> If you revoke a member’s disk store, do not restart that member with its disk stores—in isolation—at a later time.

Tanzu GemFire stores information about your persisted data and prevents you from starting a member with a revoked disk store in the running system. But Tanzu 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.

Understand Cache Transactions

Understanding the operation of Tanzu GemFire transactions can help you minimize situations where the cache could become unsynchronized.

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 synchronized 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 synchronized.

Guidelines for Multi-Site Deployments

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 unsynchronized 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.

When optimizing socket-buffer sizes, use the same value for both gateway senders and gateway receivers.

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.

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.

Deactivating TCP SYN Cookies

Most default Linux installations use SYN cookies to protect the system against malicious attacks (such as DDOS) that flood TCP SYN packets.

This feature is not compatible with stable and busy Tanzu GemFire clusters. SYN Cookies protection gets incorrectly activated by normal Tanzu GemFire traffic, severely limiting bandwidth and new connection rates, and destroying SLAs. Security implementations should instead seek to prevent DDOS types of attacks by placing Tanzu GemFire server clusters behind advanced firewall protection.

To deactivate SYN cookies permanently:

1. Edit the /etc/sysctl.conf file to include the following line:

   ```
   net.ipv4.tcp_syncookies = 0
   ```

   Setting this value to zero deactivate SYN cookies.

2. Reload sysctl.conf:

   ```
   sysctl -p
   ```

Logging

This topic links to documentation about logging in VMware Tanzu GemFire.

Comprehensive logging messages help you confirm system configuration and debug problems in configuration and code.

- How Tanzu GemFire Logging Works

  VMware Tanzu GemFire uses Apache Log4j 2 as the basis for its logging system.

- Understanding Log Messages and Their Categories

  System logging messages typically pertain to startup; logging management; connection and system membership; distribution; or cache, region, and entry management.

- 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.

- Configuring Logging

  You configure logging in a member's gemfire.properties or at startup with gfsh.

- Configuring Log4j 2 for VMware GemFire

  Basic Tanzu GemFire logging configuration is configured using the gemfire.properties file. This topic is intended for advanced users who need increased control over logging due to integration with third-party libraries.

How GemFire Logging Works

This topic explains how VMware Tanzu GemFire logging works.

Tanzu GemFire leverages the Apache Log4j 2 API and Core libraries as the foundation of its logging system. The Log4j 2 API, a widely-used and robust logging framework, is utilized by all Tanzu GemFire classes to produce log statements. On the backend, Log4j 2 Core serves as the logging implementation, capable of handling output from various logging APIs. Tanzu GemFire extends this functionality with three custom Log4j 2 Appenders: GeodeConsole, GeodeLogWriter, and GeodeAlert, tailored for specific logging needs.

Tanzu GemFire has been tested with Log4j version 2.18.0 and requires the log4j-api-2.18.0.jar and log4j-core-2.18.0.jar files to be included in the classpath. These JARs are distributed in the <path-to-product>/lib directory and are also bundled within the corresponding *-dependencies.jar convenience libraries.

Custom Appenders in Tanzu GemFire

Tanzu GemFire provides two key custom Log4j 2 appenders for specialized logging functionality:

1. AlertAppender: This appender generates Tanzu GemFire alerts, which are managed through the JMX Management and Monitoring system. For more details, see Notification Federation.
2. LogWriterAppender: This appender is configured by various log-* Tanzu GemFire properties, including log-file, log-file-size-limit, and log-disk-space-limit. It handles general logging output based on these settings.

Both appenders are created and managed programmatically. Their behavior is controlled through the log-* properties and the alert level settings in the JMX Management and Monitoring system. Currently, these appenders do not support configuration through a log4j2.xml file.

Advanced Configuration

Advanced users can define a custom log4j2.xml configuration to tailor logging further. For more details, see Configuring Log4j 2 for VMware Tanzu GemFire.

Understanding Log Messages and Their Categories

This topic explains VMware Tanzu GemFire log messages and their categories.

System logging messages typically pertain to one of the following:

- Startup information: Describe the Java version, the Tanzu 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.

- Connections and system membership: Report on the arrival and departure of cluster members, including the current member, and any information related to connection activities or failures. This includes information about 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.
- Exceptions: These include standard Java exceptions and GemFire exceptions. These may originate in the current member or be propagated from another member.

Structure of a Log Message

Every logged message includes two main components:

1. Message Header (enclosed in square brackets), which consists of three parts:
    - The message level
    - The time the message was logged
    - The ID of the connection and thread that logged the message, which might be the main program or a system management process
2. Message Content, which may be a string, an exception, or an exception along with its stack trace

```
[info 2005/11/08 15:46:08.710 PST PushConsumer main nid=0x1]
Cache initialized using "file:/Samples/quickstart/xml/PushConsumer.xml".
```

Log File Name

The main log file for a Tanzu GemFire member is specified using the log-file property. This file serves as the primary log for the member and is automatically created when the application starts. If the member is not running, the file represents the most recent logging session.

By default, the main log contains all log entries for the session. However, if you configure a log-file-size-limit, Tanzu GemFire enables log rotation by splitting logs into three categories:

```
1. The **main log** continues to capture current log entries and retains the specified `log-file` name.
2. **Child logs** are created by renaming the main log once it reaches the size limit and store older entries.
3. A **metadata log** (prefixed with `meta-`) tracks operations such as startup, shutdown, and log management.
```

When log rotation is enabled, the main log is rolled into a child log as soon as it reaches the size limit, and a new main log is created. If the main log file already exists when a member starts, it is renamed to a child log to make way for a new session.

The current main log file always retains the name you specified in the log-file property. Older logs, including child log files, derive their names from the main log file and include a sequence number. If child logs are not used, the sequence number remains 00.

For standalone locators, the log file is named <locator_name>.log, based on the locator's name specified at startup. Colocated locators running within a member use the member's log file instead.

Servers started through Gfsh default to using <server_name>.log, while servers launched with an older cacheserver script use cacheserver.log.

To review recent member activities, start by checking the metadata log file (meta-<log-file>), if it exists. If no metadata log file is present, examine the main log file for the latest logging information.

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 cluster 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 cluster renames a log file, it assigns the next available sequence number in the format filename-XX-YY.extension, where XX is the log sequence number. This sequence is determined based on the numbers of existing log files and any old statistics archives. The system ensures that the next number is higher than any currently in use for either logging or statistics, keeping current log files and statistics archives aligned.

For example, if an application is logging to system.log and archiving statistics to statArchive.gfs, and it runs in a Unix directory with the following 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 are as follows:

```
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 59650 Nov 3 11:34 statArchive-04-00.gfs
-rw-rw-r-- 1 jpearson users 11308 Nov 3 11:27 system-01-00.log
-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
```

After the run, new log and statistics archive files are created, and their sequence numbers jump according to the system's naming convention. Specifically, the new files, system-04-00.log and statArchive-04-00.gfs, are assigned the next available numbers based on the existing files in the directory. In this case, the

system log file number jumps in the sequence, ensuring that both the log and statistics files remain synchronized.

Log Level

Log levels are used to categorize the severity of messages generated by your system. The higher the log level, the more urgent and critical the message. These levels help you understand the severity of the issues being reported and guide troubleshooting. When diagnosing system problems, a good approach is to lower the log level (allowing more detailed messages) and try to reproduce the issue. The additional details often provide valuable insights into the root cause of the problem.

Below are the log levels, listed in order of severity from highest to lowest.

- Fatal (highest level). Indicates a serious failure that usually prevents the system from running properly. Immediate action, such as shutting down or restarting parts of the system, is often required.

  Example of a fatal message:

  ```
  [fatal 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. Indicates a significant problem that prevents a specific operation from completing, but the system may still be able to continue running.

  Example of an error message:

  ```
  [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
  ```

- Warn. Indicates a potential issue or abnormal situation that could cause problems later. In general, warn messages describe events that are of interest to end users or system managers, or that indicate potential problems in the program or system.

  Example of a warn message:

  ```
  [warn 2008/06/09 13:09:28.163 PDT <queueTimer-client> tid=0xe]
  QueueManager - Could not create a queue. No queue servers available
  ```

- Info. This is the default setting for logging. This is for informational messages, typically geared to end users and system administrators.

  Example of a info message:

  ```
  [info 2005/10/24 11:51:35.963 PDT CacheRunner main nid=0x1]
  DistributionManager straw(7368):41714 started on 192.0.2.0[10333]
  with id straw(7368):41714 (along with 0 other DMs)
  ```

- Debug. This level provides tracing information that is generally of interest to developers. It is used for the lowest volume, most important, tracing messages.

  > Generally, you should only use this level if instructed to do so by 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.

Example of a debug message:

```
[debug 2011/06/21 11:27:24.689 PDT <locatoragent_ds_w1-gst-dev04_2104> tid=0xe] SSL Configuration:
  ssl-enabled = false
```

- Trace and All. These levels are used internally and produce extensive logs that consume significant system resources. Do not use these levels unless specifically instructed by technical support.

> Tanzu GemFire no longer supports setting system properties for VERBOSE logging. To enable VERBOSE logging, see Configuring Log4j 2 for VMware GemFire

Naming, Searching, and Creating Log Files

This topic explains how to name, search and create log files in VMware Tanzu GemFire.

Log File Naming Recommendation

To make it easier to decipher logging output, it's recommended to keep each member's log files separated. For members running on the same machine, you can achieve this by starting them in different working directories while using the same relative log-file specification.

For example, in a common gemfire.properties file, you could set the following log-file:

```
log-file=./log/member.log
```

Then, start each member in a different directory, pointing to the shared properties file with this command:

```
java -DgemfirePropertyFile=<commonDirectoryPath>/gemfire.properties
```

This approach ensures that each member has its own log files within its respective working directory.

Searching the Log Files

To effectively search and analyze your logs, use the gfsh export logs command. This command exports the logs to a single location for easier access.

```
gfsh> export logs --dir=myDir
```

After exporting the logs, you can search for specific entries by looking for lines that begin with the following strings:

- [warn
- [error
- [fatal

Creating Your Own Log Messages

In addition to the system logs, you can add your own application logs from your Java code. For information about adding custom logging to your applications, see the online Java documentation for the org.apache.geode.LogWriter interface. Both system and application logging is output and stored according to your logging configuration settings.

Configuring Logging

This topic explains how to configure VMware Tanzu GemFire logging.

You configure logging in a member's gemfire.properties or at startup with gfsh.

Before you begin, review Basic Configuration and Programming on how to configure cluster and cache properties.

1. Run a time synchronization service such as NTP on all Tanzu 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. Look for new or unexpected warnings, errors, or severe messages. 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=info
   log-file=
   log-file-size-limit=0
   log-disk-space-limit=0
   ```

   You can also specify logging parameters when you start up members (either locators or servers) using the gfsh command-line utility. In addition, you can modify log file properties and log-level settings while a member is already running by using the alter runtime command.

log-level

- Set log-level. Log levels are not case-sensitive. Options are:
  - off
  - fatal (log messages with the highest level of importance)
  - error
  - warn
  - info
  - debug
  - trace
  - all

  More detailed log levels (lower on this list) include messages from all higher levels. For instance, setting the log level to warn captures warn, error, and fatal messages.

  The most detailed, lower levels in this list (trace and all) produce extensive logs that consume significant system resources. Do not use these levels unless specifically

instructed by technical support.

log-file

- 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 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.

  > Make sure each member logs to its own files. This makes the logs easier to decipher.

log-file-size-limit

- 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.

log-disk-space-limit

- 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 can create its own log file (named gfsh-0_0.log) in the directory where you run gfsh or the gfsh.bat script. By default, gfsh does not generate log files for itself. To enable gfsh logs, set the following system property to the desired log level before starting gfsh:

   ```
   export JAVA_ARGS=-Dgfsh.log-level=[fatal|error|warn|info|debug|trace]
   ```

Configuring Log4j 2 for Tanzu GemFire

This topic explains how to configure Log4J 2 for VMware Tanzu GemFire.

Basic Tanzu GemFire logging configuration is set in the gemfire.properties file. This topic is intended for advanced users who need increased control over logging due to integration with third-party libraries.

An example log4j2.xml can be located within the product distribution at $GEMFIRE/config/log4j2.xml.

To specify your own log4j2.xml configuration file, or anything else supported by Log4j 2 such as .json or .yaml, use the following flag when starting up your JVM or Tanzu GemFire member:

```
-Dlog4j.configurationFile=<location-of-your-file>
```

If the Java system property log4j.configurationFile is specified, then Log4j will not use the log4j2.xml included in gemfire-log4j-<version>.jar.

Using Different Front-End Logging APIs to Log to Log4j2

You can also configure Log4j 2 to work with various popular and commonly used logging APIs. To obtain and configure the most popular front-end logging APIs to log to Log4j 2, see Logging Services on the Apache Log4j 2 website.

For example, if you are using:

- Commons Logging, download “Commons Logging Bridge” (log4j-jcl-2.18.0.jar)
- SLF4J, download “SLFJ4 Binding” (log4j-slf4j-impl-2.18.0.jar)
- java.util.logging, download the “JUL adapter” (log4j-jul-2.18.0.jar)

For more examples, see Frequently Asked Questions on the Apache Log4j 2 website.

All three of the above JAR files are in the full distribution of Log4J 2.18.0 which can be downloaded from Download Apache Log4j™ 2 on the Apache Log4j 2 website. Download the appropriate bridge, adapter, or binding JARs to ensure that Tanzu GemFire logging is integrated with every logging API used in various third-party libraries or in your own applications.

> VMware Tanzu GemFire has been tested with Log4j 2.18.0. As newer versions of Log4j 2 come out, you can find 2.18.0 under Previous Releases on that page.

Customizing Your Own log4j2.xml File

Advanced users may want to move away entirely from setting log-* gemfire properties and instead specify their own log4j2.xml using -Dlog4j.configurationFile.

Custom Log4j 2 configuration in Tanzu GemFire comes with some caveats and notes:

- Do not use "monitorInterval=" in your log4j2.xml file, because doing so can have significant performance impact. This setting instructs Log4j 2 to monitor the log4j2.xml config file at runtime and automatically reload and reconfigure if the file changes.
- Tanzu GemFire’s default log4j2.xml specifies status=“FATAL” because Log4j 2’s StatusLogger generates warnings to standard out at ERROR level anytime Tanzu GemFire stops its AlertAppender or LogWriterAppender. Tanzu GemFire uses a lot of concurrent threads that are executing code with log statements; these threads may be logging while the Tanzu GemFire appenders are being stopped.

- Tanzu GemFire's default log4j2.xml specifies shutdownHook="disable" because Tanzu GemFire has a shutdown hook which disconnects the DistributedSystem and closes the Cache, which is executing the code that performs logging. If the Log4J2 shutdown hook stops logging before Tanzu GemFire completes its shutdown, Log4j 2 will attempt to start back up. This restart in turn attempts to register another Log4j 2 shutdown hook which fails resulting in a FATAL level message logged by Log4j 2.
- The GEODE_VERBOSE marker can be used to enable additional verbose log statements at TRACE level. For more information about Log4J2 Markers, see Markers on the Apache Log4j 2 website. Many log statements are enabled simply by enabling DEBUG or TRACE. However, even more log statements can be further enabled by using MarkerFilter to accept GEODE_VERBOSE. The default Tanzu GemFire log4j2.xml deactivate GEODE_VERBOSE with this line:

  ```
  <MarkerFilter marker="GEODE_VERBOSE" onMatch="DENY" onMismatch="NEUTRAL"/>
  ```

  You can enable the GEODE_VERBOSE log statements by changing onMatch="DENY" to onMatch="ACCEPT". Typically, it is more useful to simply enable DEBUG or TRACE on certain classes or packages instead of for the entire Tanzu GemFire product. However, this setting can be used for internal debugging purposes if all other debugging methods fail.
- The usage of filters can have an impact on performance. Tanzu GemFire has some logging optimizations that are deactivated when filters are used (except for the GEODE_VERBOSE marker mentioned previously).
- Geode's custom Log4j 2 Appenders can be used in a custom log4j2.xml. Specify GeodeConsole, GeodeLogWriter, and or GeodeAlert in the Appenders section:

  ```
  <Properties>
    <Property name="gemfire-pattern">[%level{lowerCase=true} %date{yyyy/MM/dd HH:mm:ss.SSS z} <%thread> tid=%
  hexTid] %message%n%throwable%n</Property>
  </Properties>

  <Appenders>
    <GeodeConsole name="STDOUT" target="SYSTEM_OUT">
      <PatternLayout pattern="${gemfire-pattern}"/>
    </GeodeConsole>
    <GeodeLogWriter name="LOGWRITER">
      <PatternLayout pattern="${gemfire-pattern}"/>
    </GeodeLogWriter>
    <GeodeAlert name="ALERT"/>
  </Appenders>
  ```

  (Note that GeodeAlert does not use a PatternLayout)

  See the default log4j2.xml in your GemFire distribution's config directory as a reference for creating a custom configuration.

Statistics

This topic links to documentation about gathering and using statistics in VMware Tanzu GemFire.

Every application and server in a cluster can access statistical data about VMware Tanzu GemFire operations. You can configure the gathering of statistics by using the alter runtime command of gfsh or in the gemfire.properties file to facilitate system analysis and troubleshooting.

- How Statistics Work

Each application or cache server that joins the cluster can collect and archive statistical data for analyzing system performance.

- Transient Region and Entry Statistics

  For replicated, distributed, and local regions, Tanzu GemFire provides a standard set of statistics for the region and its entries.

- Application-Defined and Custom Statistics

  Tanzu GemFire includes interfaces for defining and maintaining your own statistics.

- Configuring and Using Statistics

  You configure statistics and statistics archiving in several different ways.

- Viewing Archived Statistics

How Statistics Work

This topic explains how statistics work in VMware Tanzu GemFire.

Each application or cache server that joins the cluster can collect and archive statistical data for analyzing system performance.

Tanzu GemFire statistics can be enabled for a cluster, for an application, for a server, or for a region. Statistics gathered for a cluster, an application, or a cache server are saved to a file and can be archived, whereas region statistics are transient and accessible only through the API.

Set the configuration attributes that control cluster, application, or cache statistics collection in gfsh or in the gemfire.properties configuration file. You can also collect your own application defined statistics.

When Java applications and servers join a cluster, they can be configured via the cluster configuration service to enable statistics sampling and whether to archive the statistics that are gathered.

> Tanzu GemFire statistics use the Java System.nanoTimer for nanosecond timing. This method provides nanosecond precision, but not necessarily nanosecond accuracy. For more information, see the online Java documentation for System.nanoTimer for the JRE you are using with Tanzu GemFire.

Statistics sampling provides valuable information for ongoing system tuning and troubleshooting. Sampling statistics at the default sample rate does not impact overall cluster performance. We recommend enabling statistics sampling in production environments.

Transient Region and Entry Statistics

This topic explains transient region and entry statistics in VMware Tanzu GemFire.

For replicated, distributed, and local regions, Tanzu GemFire provides a standard set of statistics for the region and its entries.

GemFire gathers these statistics when the --enable-statistics parameter of the create region command of gfsh is set to true or in cache.xml the region attribute statistics-enabled is set to true.

> Unlike other GemFire statistics, these region and entry statistics are not archived and cannot be charted.

Enabling these statistics requires extra memory per entry. See Memory Requirements for Cached Data.

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

This topic discusses application-defined and custom statistics in VMware Tanzu GemFire.

Tanzu GemFire includes interfaces for defining and maintaining your own statistics.

The Tanzu GemFire package, org.apache.geode, includes the following interfaces for defining and maintaining your own statistics:

- StatisticDescriptor. Describes an individual statistic. Each statistic has a name and information about 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 StatisticDescriptors and provides access methods to them. The StatisticDescriptors 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, getting individual StatisticDescriptor values, and setting a callback which will recompute the statistic’s value at configured sampling intervals.
- 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. The figures below illustrate the relationships between the statistics interfaces and show the implementation of this use case.

The Statistics Interfaces

Each StatisticDescriptor contains one piece of statistical information. StatisticalDesriptor objects are collected into a StatisticsType. The StatisticsType is instantiated to create a Statistics object.

Statistics Implementation

The StatisticDescriptor objects shown here hold three pieces of statistical information about client session state. These are collected into a SessionStateStats StatisticsType. With this type, the server creates a Statistics object for each client that connects.

Configuring and Using Statistics

This topic explains how to configure VMware Tanzu GemFire statistics and statistics archiving in several ways.

Configuring Cluster or Server Statistics

For this procedure, it is assumed that you understand Basic Configuration and Programming.

Execute the following commands to modify the cluster's configuration and enable cluster or server statistics.

```
gfsh>start locator --name=l1 --enable-cluster-configuration=true
gfsh>alter runtime --enable-statistics=true --statistic-archive-file=myStatisticsArchiveFile.gfs
```

> The alter runtime --enable-statistics parameter is deprecated as it will be enabled by default. It will be removed in upcoming releases.

Note that setting statistic-archive-file to /dev/null still calculates statistics, but they are not archived to a file.

You can also configure sample rate and the file name of your statistic archive files. See alter runtime for more command options.

Alternatively, if you are not using the cluster configuration service, configure gemfire.properties for the statistics monitoring and archiving that you need:

1. Enable statistics gathering for the cluster. This is required for all other statistics activities:

   ```
   statistic-sampling-enabled=true
   statistic-archive-file=myStatisticsArchiveFile.gfs
   ```

   > The statistic-sampling-enabled property is deprecated as it will be enabled by default. It will be removed in upcoming releases.
   >
   > Statistics sampling at the default sample rate (1000 milliseconds) does not impact system performance and is recommended in production environments for troubleshooting.

2. Change the statistics sample rate as needed. Example:

   ```
   statistic-sampling-enabled=true
   statistic-archive-file=myStatisticsArchiveFile.gfs
   statistic-sample-rate=2000
   ```

3. 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
   ```

4. If you need time-based statistics, enable that. Time-based statistics require statistics sampling and archiving. This setting also enables Micrometer meters of type timer. Example:

   ```
   statistic-sampling-enabled=true
   statistic-archive-file=myStatisticsArchiveFile.gfs
   enable-time-statistics=true
   ```

If these statistics are enabled, you can access archived statistics through the gfsh show metrics command.

Configure Transient Region and Entry Statistics

Enable transient region and entry statistics gathering on the regions where you need them. This configuration is distinct from the enabling of cluster or server statistics.

gfsh example:

```
gfsh>create region --name=myRegion --type=REPLICATE --enable-statistics=true
```

cache.xml example:

```
<region name="myRegion" refid="REPLICATE">
  <region-attributes statistics-enabled="true">
  </region-attributes>
</region>
```

API example:

Region and entry statistics are not archived and can be accessed only 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());
 }
```

Configure Custom Statistics

Create and manage any custom statistics that you need through cache.xml and the API.

cache/cluster.xml example:

```
// Create custom statistics
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE statistics PUBLIC
   "-//Example Systems, Inc.//Example Statistics Type//EN"
   "http://www.example.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>
```

API example:

```
// Update custom stats through the API
this.samplerStats.incInt(this.sampleCountId, 1);
this.samplerStats.incLong(this.sampleTimeId, nanosSpentWorking / 1000000);
```

Controlling the Size of Archive Files

You can specify limits on the archive files for statistics using the gfsh alter runtime command. These are the areas of control:

- Archive File Growth Rate.
    - The --statistic-sample-rate parameter controls how often samples are taken, which affects the speed at which the archive file grows.
    - The --statistic-archive-file parameter 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 --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.

  > If you modify the value of --archive-file-size-limit while the cluster is running, the new value does not take effect until the current archive becomes inactive (that is, when a new archive is started).

- Maximum Size of All Archive Files. The --archive-disk-space-limit parameter 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.

> If you modify the value of --archive-disk-space-limit while the cluster is running, the new value does not take effect until the current archive becomes inactive.

Viewing Archived Statistics

This topic discusses how to view archived historical data in VMware Tanzu GemFire.

When sampling and archiving are enabled, you can examine archived historical data to help diagnose performance problems. Study statistics in archive files by using the gfsh show metrics command. Additionally, the gfsh show statistics command can show point-in-time statistics from one or more members. You may also wish to use a separate statistics display utility, such as the Visual Statistics Display tool (VSD) provided with Tanzu GemFire. The VSD tool reads the sampled statistics and produces graphical displays for analysis.

Troubleshooting and System Recovery

This topic provides links to VMware Tanzu GemFire troubleshooting and system recovery documentation. The linked documentation provide strategies for handling common errors and failure situations.

- Producing Artifacts for Troubleshooting

  There are several types of files that are critical for troubleshooting.

- Diagnosing System Problems

  This section provides possible causes and suggested responses for system problems.

- 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.

- Handling Forced Cache Disconnection Using Auto-reconnect

  A Tanzu GemFire member may be forcibly disconnected from a cluster if the member is unresponsive for a period of time, or if a network partition separates one or more members into a

group that is too small to act as the cluster.

- 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 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.

- Recovering from ConflictingPersistentDataExceptions

  A ConflictingPersistentDataException while starting up persistent members indicates that you have multiple copies of some persistent data, and Tanzu GemFire cannot determine which copy to use.

- Preventing and Recovering from Disk Full Errors

  It is important to monitor the disk usage of Tanzu GemFire members. If a member lacks sufficient disk space for a disk store, the member attempts to shut down the disk store and its associated cache, and logs an error message. A shutdown due to a member running out of disk space can cause loss of data, data file corruption, log file corruption and other error conditions that can negatively impact your applications.

- 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.

- Log Messages and Solutions

  This section provides strategies for responding to a variety of system log messages.

Producing Artifacts for Troubleshooting

This topic describes VMware Tanzu GemFire troubleshooting artifacts and explains how to produce them.

There are several types of files that are critical for troubleshooting.

Tanzu GemFire logs and statistics are the two most important artifacts used in troubleshooting. In addition, they are required for Tanzu 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_UserProfileName_\AppData\Local\Microsoft\Windows\WER\ReportArchive

- C:\Users_UserProfileName_\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 clusters involved. If you are running a client/server architecture, create log files for the clients.

> 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
   ```

   > Running with the log level at fine can impact system performance and fill up your disk.

4. Enable statistics gathering for the cluster 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
   ```

   > The statistic-sampling-enabled property and the alter runtime --enable-statistics parameter are deprecated as statistic collection will be enabled by default. It will be removed in upcoming releases.
   >
   > 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 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 topic provides information about and suggestions for diagnosing and resolving VMware Tanzu GemFire system problems.

- Locator does not start
- Application or cache server process does not start
- Application or cache server does not join the cluster
- Member process seems to hang
- Member process does not read settings from the gemfire.properties file
- Cache creation fails - must match schema definition root
- Cache is not configured properly
- Unexpected results for keySetOnServer and containsKeyOnServer
- Data operation returns PartitionOfflineException
- Entries are not being evicted or expired as expected
- Cannot find the log file
- OutOfMemoryError
- PartitionedRegionDistributionException
- PartitionedRegionStorageException
- Application crashes without producing an exception
- Timeout alert
- Member produces SocketTimeoutException
- Member logs ForcedDisconnectException, Cache and DistributedSystem forcibly closed
- Members cannot see each other
- One part of the cluster cannot see another part
- Data distribution has stopped, although member processes are running
- Distributed-ack operations take a very long time to complete
- Slow system performance
- Cannot get Windows performance data
- Java applications on 64-bit platforms hang or use 100% CPU

Locator does not start

Invocation of a locator with gfsh fails with an error like this:

```
Starting a GemFire Locator in C:\devel\gfcache\locator\locator
The Locator process terminated unexpectedly with exit status 1.
Please refer to the log file in C:\devel\gfcache\locator\locator for full details.
Exception in thread "main" java.lang.RuntimeException: An IO error occurred while
starting a Locator in C:\devel\gfcache\locator\locator on 192.0.2.0[10999]: Network is
unreachable; port (10999) is not available on 192.0.2.0.
...
```

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 this cluster, 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 for more information.

Application or cache server process does not start

If the process tries to start and then silently disappears, on Windows this indicates a memory problem.

Response:

- On a Windows host, decrease the maximum 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.
- If this does not work, try rebooting.

Application or cache server does not join the cluster

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 Tanzu GemFire system are located outside the LAN, check whether the firewall is blocking communication. Tanzu 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.
- Can not connect to locator (when using TCP for discovery).
  - 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, although member processes are running.

    - 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 Topology and Communication General Concepts for more information about using bind addresses.

- Wrong version of Tanzu GemFire. A version mismatch can cause the process to hang or crash. Check the software version with the gemfire version command.

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. 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 cluster 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 cannot 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. Tanzu GemFire looks for a gemfire.properties file in the current working directory, the home directory, and the class path, 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 schema definition root

System member startup fails with an error like one of these:

```
Exception in thread "main" org.apache.geode.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" org.apache.geode.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".
```

Tanzu GemFire declarative cache creation uses one of two 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 XML namespace and schema definition.

For peers and servers:

```
<?xml version="1.0" encoding="UTF-8"?>
<cache
   xmlns="http://geode.apache.org/schema/cache"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cache-1.0.xsd"
   version="1.0">
...
</cache>
```

For clients:

```
<?xml version="1.0" encoding="UTF-8"?>
<client-cache
   xmlns="http://geode.apache.org/schema/cache"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cache-1.0.xsd"
   version="1.0">
...
</client-cache>
```

Cache is not 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 is not reading the cache configuration file. Check that the name and location of the cache configuration file match those configured in the cache-xml-file attribute in gemfire.properties. If they match, the process may not be reading gemfire.properties. See Member process does not read settings from the gemfire.properties file.
- Regions without data—If the cache starts with regions, but no data, this process may not have joined the correct cluster. Check the log file for messages that indicate other members. If you do not see any, the process may be running alone in its own cluster. In a process that is clearly part of the correct cluster, regions without data may indicate an implementation design error.

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 Options.

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:

```
org.apache.geode.cache.persistence.PartitionOfflineException:
Region /__PR/_B__root_partitioned__region_7 has persistent data that is no
longer online stored at these locations:
[/192.0.2.1:/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.

Entries are not being evicted or expired as expected

Check these possible causes.

- Transactions—Entries that are due to be expired 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.
- 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 and Expiration.

Cannot 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.

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.
- You may need to lower the thread stack size. The default thread stack size is 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.

PartitionedRegionDistributionException

The org.apache.geode.cache.PartitionedRegionDistributionException appears when Tanzu 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 Tanzu GemFire processes.
- If you keep seeing PartitionedRegionDistributionException, you should evaluate whether you need to start more members.

PartitionedRegionStorageException

The org.apache.geode.cache.PartitionedRegionStorageException appears when Tanzu GemFire cannot 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 Tanzu 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.

Response: Control memory use by setting entry limits for the regions.

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 has not responded. The alert is logged in the sender's log as a warning.

A timeout alert can be considered normal.

Response:

- If you see numerous timeouts and you have not 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 and server 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. For a client/server configuration, adjust the "read-timeout" value as described in <pool> or use the org.apache.geode.cache.client.PoolFactory.setReadTimeout method.

Member logs ForcedDisconnectException, Cache and DistributedSystem forcibly closed

A cluster 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 Tanzu GemFire log file for the member shows a ForcedDisconnectException with the message

```
This member has been forced out of the cluster 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 about using bind addresses, see Topology and Communication General Concepts.
- Check that all the applications and cache servers are using the same locator address.

One part of the cluster 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.

Also see Understanding and Recovering from Network Outages.

Data distribution has stopped, although member processes are running

Suspect a problem with the network, or the locator.

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.

- Check your network monitoring tools to see whether the network is down or flooded.
- Create a watchdog daemon or service on each locator host to restart the locator process when it stops
- 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 cluster 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 cluster 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.

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 about alleviating this problem, see Slow distributed-ack Messages.

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.

cannot get Windows performance data

Attempting to run performance measurements for Tanzu GemFire on Windows can produce this error message:

```
cannot 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 Tanzu 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 Tanzu 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)"
```

System Failure and Recovery

This topic describes VMware Tanzu GemFire alerts and appropriate responses to various kinds of system failures. Use this topic to plan a strategy for data recovery.

If a system member withdraws from the cluster involuntarily because the member, host, or network fails, the other members automatically adapt to the loss and continue to operate. The cluster 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.
- Whether the region is configured for LRU-based eviction.
- 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 about 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.

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 cluster 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 cluster 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: org.apache.geode.ForcedDisconnectException:
There are no processes eligible to be group membership coordinator
(last coordinator left view)
```

Description:

Network partition detection is enabled, 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: org.apache.geode.ForcedDisconnectException:
There are no processes eligible to be group membership coordinator
(all eligible coordinators are suspect)
```

Description:

Network partition detection is enabled, 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: org.apache.geode.ForcedDisconnectException:
Unable to contact any locators and network partition detection is enabled
```

Description:

Network partition detection is enabled, 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: org.apache.geode.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, VMware recommends that you 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: org.apache.geode.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 cluster 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.

Disk Access Exceptions

Alert:

```
A DiskAccessException has occurred while writing to the disk for region <region-name>.
The cache will be closed.  For Region: <region-name>: Failed writing key
to <disk-store-name>
```

or

```
A DiskAccessException has occurred while writing to the disk for region <region-name>.
The cache will be closed.
For DiskStore: <disk-store-name>: Could not schedule asynchronous write because
the flusher thread had been terminated
```

Description:

A write was prevented by an underlying disk issue, such as a full disk.

The first alert form is reported when disk writes are synchronous (disk-synchronous=true), and the second form is reported when disk writes are asynchronous (disk-synchronous=false).

In either case, the member shuts down when an operation attempts to update the disk store.

Response:

You must address the underlying disk issue and restart the server. See Preventing and Recovering from Disk Full Errors for suggestions.

How Data is Recovered From Persistent Regions

A persistent region is one whose contents (keys and values) can be restored from disk. Upon restart, data recovery of a persistent region always recovers keys. Under the default behavior, the region is regarded as ready for use when the keys have been recovered.

The default behavior for restoring values depends on whether the region was configured with an LRU-based eviction algorithm:

- If the region was not configured for LRU-based eviction, the values are loaded asynchronously on a separate thread. The assumption here is that all of the stored values will fit into the space allocated for the region.
- If the region was configured for LRU-based eviction, the values are not loaded. Each value will be retrieved only when requested. The assumption here is that the values resident in the region plus any evicted values might exceed the space allocated for the region, possibly resulting in an OutOfMemoryException during recovery. Note: Recovered values do not contain usage history—LRU history is reset at recovery time.

This default behavior works well under most circumstances. For special cases, three Java system properties allow the developer to modify the recovery behavior for persistent regions:

- gemfire.disk.recoverValues

  Default = true, recover values for non-LRU regions. Enables the possibility of recovering values for LRU regions (with the setting of an additional property). If false, recover only keys, do not recover values. The false setting disallows value recovery for LRU regions as well as non-LRU regions.

  How used: When true, recovery of the values “warms up” the cache so data retrievals will find their values in the cache, without causing time consuming disk accesses. When false, shortens recovery time so the system becomes available for use sooner, but the first retrieval on each key will require a disk read.

- gemfire.disk.recoverLruValues

Default = false, do not recover values for a region configured with LRU-based eviction. If true, recover all of the LRU region's values. Note: gemfire.disk.recoverValues must also be true for this property to take effect.

How used: When false, shortens recovery time for an LRU-configured region by not loading values. When true, restores data values to the cache. As stated above, LRU history is not recoverable, and recovering values for a region configured with LRU-based eviction incurs some risk of exceeding allocated memory.

- gemfire.disk.recoverValuesSync

  Default = false, recover values by an asynchronous background process. If true, values are recovered synchronously, and recovery is not complete until all values have been retrieved. Note: gemfire.disk.recoverValues must also be true for this property to take effect.

  How used: When false, allows the system to become available sooner, but some time must elapse before all values have been read from disk into cache memory. Some key retrievals will require disk access, and some will not. When true, prolongs restart time, but ensures that when available for use, the cache is fully populated and data retrieval times will be optimal.

Handling Forced Cache Disconnection Using Auto-reconnect

This topic explains how to use auto-reconnect in VMware Tanzu GemFire to handle forced cache disconnection.

A Tanzu GemFire member may be forcibly disconnected from a Tanzu GemFire cluster if the member is unresponsive for a period of time, or if a network partition separates one or more members into a group that is too small to act as the cluster.

How the Auto-reconnection Process Works

After being disconnected from a cluster, a Tanzu GemFire member shuts down and, by default, automatically restarts into a "reconnecting" state, while periodically attempting to rejoin the cluster by contacting a list of known locators. If the member succeeds in reconnecting to a known locator, the member rebuilds its view of the cluster from existing members and receives a new distributed member ID.

If the member cannot connect to a known locator, the member will then check to see if it itself is a locator (or hosting an embedded locator process). If the member is a locator, then the member does a quorum-based reconnect; it will attempt to contact a quorum of the members that were in the membership view just before it became disconnected. If a quorum of members can be contacted, then startup of the cluster is allowed to begin. Since the reconnecting member does not know which members survived the network partition event, all members that are in a reconnecting state will keep their membership ports open and respond to ping requests.

Membership quorum is determined using the same member weighting system used in network partition detection. See Membership Coordinators, Lead Members, and Member Weighting.

Note that when a locator is in the reconnecting state, it provides no discovery services for the cluster.

The default settings for reconfiguration of the cache once reconnected assume that the cluster configuration service has a valid (XML) configuration. This will not be the case if the cluster was configured using API calls. To handle this case, either deactivate auto-reconnect by setting the property to

```
disable-auto-reconnect = true
```

or, deactivate the cluster configuration service by setting the property to

```
enable-cluster-configuration = false
```

After the cache has reconnected, applications must fetch a reference to the new Cache, Regions, DistributedSystem and other artifacts. Old references will continue to throw cancellation exceptions like CacheClosedException(cause=ForcedDisconnectException).

See the Tanzu GemFire DistributedSystem and Cache Java API documentation for more information.

Managing the Auto-reconnection Process

By default a Tanzu GemFire member will try to reconnect until it is told to stop by using the DistributedSystem.stopReconnecting() or Cache.stopReconnecting() method. You can deactivate automatic reconnection entirely by setting disable-auto-reconnect Tanzu GemFire property to “true.”

You can use DistributedSystem and Cache callback methods to perform actions during the reconnect process, or to cancel the reconnect process if necessary.

The DistributedSystem and Cache API provide several methods you can use to take actions while a member is reconnecting to the cluster:

- DistributedSystem.isReconnecting() returns true if the member is in the process of reconnecting and recreating the cache after having been removed from the system by other members.
- DistributedSystem.waitUntilReconnected(long, TimeUnit) waits for a period of time, and then returns a boolean value to indicate whether the member has reconnected to the DistributedSystem. Use a value of -1 seconds to wait indefinitely until the reconnect completes or the member shuts down. Use a value of 0 seconds as a quick probe to determine if the member has reconnected.
- DistributedSystem.getReconnectedSystem() returns the reconnected DistributedSystem.
- DistributedSystem.stopReconnecting() stops the reconnection process and ensures that the DistributedSystem stays in a disconnected state.
- Cache.isReconnecting() returns true if the cache is attempting to reconnect to a cluster.
- Cache.waitUntilReconnected(long, TimeUnit) waits for a period of time, and then returns a boolean value to indicate whether the DistributedSystem has reconnected. Use a value of -1 seconds to wait indefinitely until the reconnect completes or the cache shuts down. Use a value of 0 seconds as a quick probe to determine if the member has reconnected.
- Cache.getReconnectedCache() returns the reconnected Cache.
- Cache.stopReconnecting() stops the reconnection process and ensures that the DistributedSystem stays in a disconnected state.

Operator Intervention

You may need to intervene in the auto-reconnect process if processes or hardware have crashed or are otherwise shut down before the network connection is healed. In this case the members in a “reconnecting” state will not be able to find the lost processes and will not rejoin the system until they can contact a locator.

Recovering from Application and Cache Server Crashes

This topic explains how to recover VMware Tanzu GemFire 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.

- Recovering from Crashes with a Client/Server Configuration

  In a client/server configuration, you first make the server available as a member of a cluster again, and then restart clients as quickly as possible. The client recovers its data from its servers through normal operation.

Recovering from Crashes with a Peer-to-Peer Configuration

This topic explains how to recover VMware Tanzu GemFire 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 is configuring a distributed region. For example:

```
<region-attributes scope="distributed-no-ack">
```

If the file includes either of the following lines, it is 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 with the remaining copies of the data. You may need to perform recovery actions depending on how many members you have lost and how you have configured redundancy in your system.

By default, Tanzu 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.

To recover, start a replacement member. The new member regenerates the lost copies and returns them to the configured redundancy level.

> Make sure the replacement member has at least as much local memory as the old one—the local-max-memory configuration setting must be the same or larger. Otherwise, you can get into a situation where some entries have all their redundant copies but others do not. In addition, until you have restarted a replacement member, any code that attempts to create or update data mapped to partition region bucket copies (primary and secondary) that have been lost can result in an exception. (New transactions unrelated to the lost data can fail as well simply because they happen to map to– or "resolve" to– a common bucketId).

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 only 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.

Recovery Without Data Redundancy

If a member crashes and there are no redundant copies, any logic that tries to interact with the bucket data is *blocked* until the primary buckets are restored from disk. (If you do not have persistence enabled, Tanzu GemFire will reallocate the buckets on any available remaining nodes, however you will need to recover any lost data using external mechanisms.)

To recover, restart the member. The application returns to active work and automatically begins to create new data.

If the members with the relevant disk stores cannot be restarted, then you will have to revoke the missing disk stores manually using gfsh. See revoke missing-disk-store.

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)-192.0.2.0>
tid=0x1786] Unable to find sufficient members to host a bucket in the partitioned region.
Region name = /partitionedregion Current number of available data stores: 1 number
successfully allocated = 1 number needed = 2 Data stores available:
[pippin(21944):41927/42712] Data stores successfully allocated:
[pippin(21944):41927/42712] Consider starting another member
```

The following alert [EXCEPTION-1] (warning) and exception is generated when, after creating 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)-192.0.2.0>
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:

```
org.apache.geode.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 synchronized 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. 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, last modified time is persisted from before the member exit or crash. For information about how this might affect the region data, see Expiration.

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 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 may also have been stopped. Recreate the region and restart the cache servers.

Recovering from Crashes with a Client/Server Configuration

This topic explains how to recover VMware Tanzu GemFire from crashes with a client/server configuration.

In a client/server configuration, you first make the server available as a member of a cluster 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 Configuring Client/Server Event Messaging for information about durable messaging.

Recovering from Server Failure

Recovery from server failure has two parts: the server recovers as a member of a cluster, then its clients recover its services.

When servers fail, their own recovery is carried out as for any member of a cluster as described in Recovering from Crashes with a Peer-to-Peer Configuration.

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.

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.

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. Cache modification events for entries in which the client has registered durable interest are saved.

If you have a durable client configured to connect to multiple servers, keep in mind that Tanzu 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

This topic explains how to recover VMware Tanzu GemFire 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 cluster unexpectedly.

Recovery Procedure

To recover from a machine crash:

1. Determine which processes run on this machine.
2. Reboot the machine.
3. If a Tanzu GemFire locator runs here, start it first.

   > At least one locator must be running before you start any applications or cache servers.

4. Start the applications and cache servers in the usual order.

If you have to move a locator process to a different machine, the locator is not useful until you update the locators list in the gemfire.properties file and restart all the applications and cache servers in the cluster. If other locators are running, however, you do not 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, Tanzu 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.

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 Tanzu GemFire cluster.

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.

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 and Data Recovery for Distributed Regions 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.

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.

Recovering from ConfictingPersistentDataExceptions

This topic explains how to recover VMware Tanzu GemFirefrom a ConflictingPersistentDataException.

A ConflictingPersistentDataException while starting up persistent members indicates that you have multiple copies of some persistent data, and Tanzu GemFire cannot determine which copy to use.

Normally, Tanzu GemFire uses metadata to determine automatically which copy of persistent data to use. Along with the region data, each member persists a list of other members that are hosting the region and whether their data is up to date. A ConflictingPersistentDataException happens when two members compare their metadata and find that it is inconsistent. The members either do not know about each other, or they both think the other member has stale data.

The following sections describe scenarios that can cause ConflictingPersistentDataExceptions in Tanzu GemFire and how to resolve the conflict.

Independently Created Copies

Trying to merge two independently created clusters into a single cluster will cause a ConflictingPersistentDataException.

There are a few ways to end up with independently created systems.

- Create two different clusters by having members connect to different locators that are not aware of each other.
- Shut down all persistent members and then start up a different set of brand new persistent members.

Tanzu GemFire will not automatically merge independently created data for the same region. Instead, you need to export the data from one of the systems and import it into the other system. See the section Cache and Region Snapshots for instructions on how to export data from one system and import it into another.

Starting New Members First

Starting a brand new member that has no persistent data before starting older members with persistent data can cause a ConflictingPersistentDataException.

One accidental way this can happen is to shut the system down, add a new member to the startup scripts, and start all members in parallel. By chance, the new member may start first. The issue is that the new member will create an empty, independent copy of the data before the older members start up. Tanzu GemFire will be treat this situation like the Independently Created Copies case.

In this case the fix is simply to move aside or delete the persistent files for the new member, shut down the new member and then restart the older members. When the older members have fully recovered, then restart the new member.

A Network Failure Occurs and Network Partitioning Detection is Deactivated

When enable-network-partition-detection is set to the default value of true, Tanzu GemFire will detect a network partition and shut down unreachable members to prevent a network partition (“split brain”) from

occurring. No conflicts should occur when the system is healed.

However if enable-network-partition-detection is false, Tanzu GemFire will not detect the network partition. Instead, each side of the network partition will end up recording that the other side of the partition has stale data. When the partition is healed and persistent members are restarted, the members will report a conflict because both sides of the partition think the other members are stale.

In some cases it may be possible to choose between sides of the network partition and just keep the data from one side of the partition. Otherwise you may need to salvage data and import it into a fresh system.

Salvaging Data

If you receive a ConflictingPersistentDataException, you will not be able to start all of your members and have them join the same cluster. You have some members with conflicting data.

First, see if there is part of the system that you can recover. For example if you just added some new members to the system, try to start up without including those members.

For the remaining members you can extract data from the persistent files on those members and import the data.

To extract data from the persistent files, use the gfsh export offline-disk-store command.

```
gfsh> export offline-disk-store --name=MyDiskStore --disk-dirs=./mydir --dir=./outputdir
```

This will produce a set of snapshot files. Those snapshot files can be imported into a running system using:

```
gfsh> import data --region=/myregion --file=./outputdir/snapshot-snapshotTest-test0.gfd --member=server1
```

Preventing and Recovering from Disk Full Errors

This topic explains how to prevent and recover from disk full errors when using VMware Tanzu GemFire.

VMware recommends that you monitor the disk usage of Tanzu GemFire members. If a member lacks sufficient disk space for a disk store, the member shuts down.

After you make sufficient disk space available to the member, you can restart the member.

You can prevent disk full errors using the following techniques:

- If you are using ext4 file system, we recommend that you pre-allocate disk store files and disk store metadata files. Pre-allocation reserves disk space for these files and leaves the member in a healthy state when the disk store and regions are shut down, allowing you to restart the member once sufficient disk space has been made available. Pre-allocation is enabled by default.
- Configure critical usage thresholds (disk-usage-warning-percentage and disk-usage-critical-percentage) for the disk. By default, these are set to 90% for warning and 99% for errors that will shut down the cache.
- Follow the recommendations in Optimizing a System with Disk Stores for general disk management best practices.

When a disk write fails due to disk full conditions, the member is shutdown and removed from the cluster.

Recovering from Disk Full Errors

If a member of your cluster fails due to a disk full error condition, add or make additional disk capacity available and attempt to restart the member normally. If the member does not restart and there is a redundant copy of its regions in a disk store on another member, you can restore the member using the following steps:

1. Delete or move the disk store files from the failed member.
2. Use the gfsh show missing-disk-stores command to identify any missing data. You may need to manually restore this data.
3. Revoke the missing disk stores using the revoke missing-disk-store gfsh command.
4. Restart the member.

See Handling Missing Disk Stores for more information.

Understanding and Recovering from Network Outages

This topic discusses network outages with VMware Tanzu GemFire and how to recove from a network outage.

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 will not resurrect 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 clusters that cannot rejoin automatically.

What Happens During a Network Outage

One member acts in the role of membership coordinator. If that coordinator loses contact with some other members, say, due to a network problem, the coordinator treats those members as crashed.

- If the weight of the crashed members is less than half the total weight of members in the cluster before the network partition, then the coordinator will continue to operate and will distribute a new view with the crashed members removed.
- If, however, the total weight of the crashed members is greater than or equal to half the weight of members in the cluster before the network partition, then the coordinator will shut down.
- If members with weight totaling half or more of the original weight of all the cluster members survive and can communicate with each other, then they will elect a new coordinator and cluster processing can continue.
- If there is no surviving group of members with sufficient weight, then no new coordinator will be chosen.

In addition, members that have been disconnected either via network partition or due to unresponsiveness will automatically try to reconnect to the cluster unless configured otherwise. See Handling Forced Cache Disconnection Using Auto-reconnect.

Recovery Procedure

For deployments that have network partition detection or auto-reconnect deactivated, to recover from a network outage:

1. Decide which applications and cache servers to restart, based on the architecture of the cluster. 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.

Effect of Network Failure on Partitioned Regions

Both sides of the cluster 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 cluster 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 do not really know what data was lost and what was not. 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 cluster 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 cluster.

Effect of Network Failure on Persistent Regions

A network failure when using persistent regions can cause conflicts in your persisted data. When you recover your system, you will likely encounter ConflictingPersistentDataException s when members start up.

For this reason, enable-network-partition-detection must be set to true if you are using persistent regions.

For information about how to recover from ConflictingPersistentDataException errors, see Recovering from ConfictingPersistentDataExceptions.

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. 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.

Servers, like applications, are members of a cluster, 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.

Log Messages and Solutions

This topic provides explanations of VMware Tanzu GemFire log messages, including potential resolutions to errors.

Depending on how your system is configured, log files can be found in a number of locations. See Log File Locations and Naming, Searching, and Creating Log Files for more information.

above heap eviction threshold

Log Message:

```
[info 2021/03/23 16:00:13.721 EDT xxx-server01 <Notification Handler> tid=0x5d] Member:
xxx(xxx-server01:29847)<v9>:11096 above heap eviction threshold
```

Log Level: info

Category: Heap GC

Meaning:

This message requires action to remain healthy. The live objects are driving heap consumption above your threshold for collecting heap. This is not a good state, as you will either be prematurely destroying data or overflowing it to disk, which can overwhelm the disk.

Potential Resolutions:

NOTE: Tanzu GemFire eviction is not truly compatible with G1GC given how G1GC behaves and how eviction assumes that garbage will be collected.

You should consider increasing the total heap. This will increase tenured space, and potentially eliminate these messages. You can also increase your eviction-threshold percentage, but this can risk growing heap to the point where you encounter heap fragmentation issues.

below heap eviction threshold

Log Message:

```
[info 2021/03/23 16:00:43.438 EDT xxx-server01 <Notification Handler> tid=0x5d] Member:
xxx(xxx-server01:29847)<v9>:11096 below heap eviction threshold
```

Log Level: info

Category: Heap GC

Meaning:

You are now below the eviction threshold, after having been above the threshold.

Potential Resolutions:

Follow the guidance provided in the “above heap eviction threshold” message.

above heap critical threshold

Log Message:

```
[error 2020/06/23 03:43:48.796 EDT <Notification Handler1> tid=0xa4] Member:
xxx(xxx-server-2:119506)<v2>:10102 above heap critical threshold. Event generated via
polling. Used bytes: 26508001280. Memory thresholds: MemoryThresholds@[2041517395
maxMemoryBytes:26843545600, criticalThreshold:95.0,
criticalThresholdBytes:25501368320, criticalThresholdClearBytes:24964497408,
evictionThreshold:85.0, evictionThresholdBytes:22817013760,
evictionThresholdClearBytes:22280142848]
```

Log Level: error

Category: Heap GC

Meaning:

This message requires URGENT action. You are in danger of Tanzu GemFire distributed system issues where a member, or members, may be kicked out with potential major business impact. The live objects are driving heap consumption above your critical threshold, so either garbage collection is proving ineffective or your usage has increased unexpectedly, taking you to much higher levels of heap consumption. Take action immediately if you ever see this, even if you were not negatively impacted at the time.

Potential Resolutions:

If you do not already have Tanzu GemFire eviction in place, acting as a level of protection to keep heap consumption lower, consider incorporating some flavor of eviction. G1GC and other newer collectors are not really compatible with HEAP_LRU eviction, so you would need to incorporate entry count or memory-based eviction.

Generally, being above the critical threshold means that you likely need to increase your total heap. If you have the critical-threshold set relatively low given your heap size, you could consider increasing this value. Having a critical-threshold of 90%, for example, with a 30 GB heap is insufficient. This is essentially wasting 3 GB of heap acting purely as overhead protection.

VMware recommends that you set the critical-threshold to see the percentage high enough such that you have a maximum of 1 GB of overhead. This means that setting the critical-threshold to 98 would be acceptable for a 100 GB heap. If you are seeing tenured heap growth with no entry count growth over time, this is likely indicative of a leak. You will need to take heap dumps and analyze them to determine why the heap is growing. It could be only temporary, if queries are running and driving heap consumption, but this should resolve itself, since Tanzu GemFire will terminate queries and eliminate that garbage.

If you are using G1GC, it is possible that you are not setting your InitiatingHeapOccupancyPercent low enough. The default of 45 is too high, so consider trying 30% to see if the tenured heap behavior becomes more stable.

below heap critical threshold

Log Message:

```
[vm1] [error 2024/04/18 10:07:44.375 MST  <RMI TCP Connection(1)-192.168.1.247> tid=0x13]
Member: 192.168.1.247(77693)<v38>:60835 below heap critical threshold. Event generated via test.
Used bytes: 750. Memory thresholds: MemoryThresholds@[1030368304 maxMemoryBytes:1000,
criticalThreshold:90.0, criticalThresholdBytes:900, criticalThresholdClearBytes:880,
evictionThreshold:0.0, evictionThresholdBytes:0, evictionThresholdClearBytes:-20]
```

Log Level: info

Category: Heap GC

Meaning:

You are now below the critical threshold, after having been above the threshold.

Potential Resolutions:

Follow the guidance provided in the above heap critical threshold message.

Environment None

Release note None

Query execution canceled after exceeding max execution time

Log Message:

```
[info 2021/02/05 03:56:08.087 EST xxx<QueryMonitor Thread> tid=0x3d9] Query execution
canceled after exceeding max execution time 600000ms. Query String = SELECT * FROM
/xxx);isCancelled = true; Total Executions = 3045391; Total Execution Time = 0
```

Log Level: info

Category: Operations

Meaning:

The query is taking longer than the configured execution time (600,000ms, in this example). It could be a rogue query. It could also be that you lack the system resources needed to accommodate the current level of activity, including this query.

Potential Resolutions:

If this persists over time, then the query is likely taking too long independent of the current system state. To allow this query to complete, you may need to increase the configured time by setting the Tanzu GemFire system property gemfire.Cache.MAX_QUERY_EXECUTION_TIME to a higher value to allow the query sufficient time to complete. If gemfire.Cache.MAX_QUERY_EXECUTION_TIME is not set, the query will never timeout unless you are using the resource manager, in which case it will timeout in five hours. This property provides some protection against a problematic query or set of queries, but requires you to understand what is driving the query times to know how high to set gemfire.Cache.MAX_QUERY_EXECUTION_TIME.

It could be that the query did not incorporate the use of a configured index, or indexes. To gather more information, you can incorporate verbose logging for your queries by setting the Tanzu GemFire system property gemfire.Query.VERBOSE.

Query execution canceled due to memory threshold crossed in system

Log Message:

```
[warning 2018/03/02 09:33:44.516 EST xxx <ServerConnection on port 40401 Thread 24>
tid=0x1a9] Server connection from [identity(xxx(14:loner):x:x,connection=2;
port=33218]: Unexpected Exception
org.apache.geode.cache.query.QueryExecutionLowMemoryException: Query execution
canceled due to memory threshold crossed in system, memory used: 23,540,738,136
bytes.
```

Log Level: warning

Category: Operations

Meaning:

A query was canceled because some member or members have crossed the critical-threshold configured in the system. To protect the members from running out of memory, the query is terminated. The message indicates the number of <n> bytes used at the time, which is certainly more than the number of bytes equating to the critical-threshold percentage, in bytes. You should also see the “above heap critical threshold” message in some logs as well if seeing this message, to understand the problem members.

Potential Resolutions:

The root cause for the heap issues needs to be investigated. Perhaps it is simply the need for more total heap. Perhaps GC activity is not collecting garbage effectively, which happens especially with some G1GC configurations. Perhaps it is a rogue query driving much more new object activity than expected, or running too long such that the tenured heap becomes much more full than normal behavior.

You could increase the critical-threshold to some higher percentage, but that may just delay the inevitable. You could configure your regions to use the eviction-threshold, which will protect the system in many cases of hitting such high levels of heap surpassing the critical-threshold configured.

Thread warnings

For example:

There are <n> stuck threads in this node
Thread <n> is stuck
Thread <n> that was executed at <time> has been stuck for <nn> seconds

Log Message:

```
[warning 2021/04/06 00:16:51.743 EDT rtp <ThreadsMonitor> tid=0x11] There are <13>
stuck threads in this node
[warning 2021/04/06 00:17:51.737 EDT rtp <ThreadsMonitor> tid=0x11] Thread <51392> is
stuck
[warning 2021/04/06 00:17:51.738 EDT rtp <ThreadsMonitor> tid=0x11] Thread <51392>
that was executed at <06 Apr 2021 00:16:12 EDT> has been stuck for <99.119 seconds>
and number of thread monitor iteration <2>
Thread Name <poolTimer-gemfire-pool-35493>
  Thread state <WAITING>
  Waiting on <java.util.concurrent.locks.ReentrantLock$NonfairSync@cae7911>
  Owned By <Function Execution Thread-2410> and ID <50995>
  Executor Group <ScheduledThreadPoolExecutorWithKeepAlive>
  Monitored metric <ResourceManagerStats.numThreadsStuck>
  Thread Stack:   UNIQUE TO EACH CASE
```

Log Level: warning

Category: Operations

Meaning:

These messages requires URGENT action, to determine whether any issues exist. It is very possible that there are no real issues, but it is also possible this is the beginning of a major issue that could impact the entire cluster. These messages require deeper investigation.

Potential Resolutions:

If you only see this issue rarely, or only for a single iteration, it is almost certainly not an issue. The word “stuck” here may be misleading. The messages are saying that it appears that this thread has been doing the same thing for a while, so it may be stuck. Some tasks, such as taking backups, doing exports, or running a rebalance, may appear to be “stuck” when in reality they are simply doing the same thing over and over as it progresses, like moving a bucket. While it may appear that we are still moving buckets, it is probably a different bucket each time.

A key indicator that a thread is truly stuck is the number of iterations, as indicated in the “has been stuck” message above. If you know that the operation is not one that should take so long, and you see an iteration of <10> or higher, you should open a support ticket for investigation. These tickets require thread dumps, multiples, across all cache servers. If you see that <13> stuck threads in this node message, the issue is likely snowballing and starting to impact this node, and the cluster could be next.

Gather artifacts, and take action. Perhaps a bounce of members, one at a time, for members showing stuck threads, would be prudent. Identifying which member to bounce can be difficult. That said, it is often possible, by analyzing the “15 seconds have elapsed” messages in your logs. This is described more in the Seconds have elapsed message in this document.

Disconnecting old DistributedSystem to prepare for a reconnect attempt

Attempting to reconnect to the DistributedSystem. This is attempt #n

Log Message:

```
[info 2021/09/21 22:45:37.863 EDT <ReconnectThread> tid=0x7f0d] Disconnecting old
DistributedSystem to prepare for a reconnect attempt
```

Log Level: info

Category: Membership

Meaning:

These messages are related, and may require action if you are not aware of why the member has been disconnected. This is often due to some instability in the distributed system caused by either network issues or GC related pauses.

Potential Resolutions:

Examine the logs of the member that is being forced out of the system. Perhaps the member became unresponsive. Look for other logging with keywords such as “elapsed”, “wakeup”, or “heartbeat”, all relatively unique words which can be searched for to proactively find potential issues. If any of these are discovered, GC tuning is likely needed.

Unable to form a TCP/IP connection in a reasonable amount of time

Log Message:

```
[info 2021/09/03 10:31:16.311 CDT <Timer-3> tid=0x79] Performing availability check
for suspect member aaa.bbb.ccc.ddd(member:5301)<v256>:41000 reason=Unable to form a
TCP/IP connection in a reasonable amount of time
```

Log Level: info, warning, fatal : Depending on the particular situation

Category: Membership

Meaning:

This message usually coincides with the availability check logging associated with suspect members. It should be investigated further by searching for other messages that may give more indication.

This specific message, if not accompanied by other “wakeup” or “heartbeat” messages, generally indicates that a member may have crashed unexpectedly, without warning. If, however, no member has crashed, the suspect member was able to respond during suspect processing and may no longer be at risk. Still, this definitely requires action to determine if you remain vulnerable to repeated occurrences.

Potential Resolutions:

This message alone does not generally reveal how to proceed to eliminate issues. That said, a deep analysis of the logs for other significant related messages may be helpful, and following the potential resolutions for those could help to reduce or eliminate these messages.

Received Suspect Message

Log Message:

```
debug 2021/02/08 05:53:04.634 IST <member-43596> tid=0x2a] Suspecting member  XXX(server1:40875)<v13>:41004
[info 2021/02/08 05:53:04.634 IST <member-43596> tid=0x2a] No longer suspecting
192.168.240.7(ch07node5:40875)<v13>:41004
[info 2021/03/29 06:46:56.304 EDT <Geode Failure Detection thread 162> tid=0x474f0c]
received suspect message from myself for XXX(YYY-server1:15972)<v16>:40000: SOME
REASON GENERALLY PROVIDED HERE
```

Log Level: info

Category: Membership

Meaning:

This message requires action. You are in danger of having a member kicked out of the distributed system, as it was already being “suspected” of being a problem for some unknown reasons that require investigation. Continuing to see these messages indicates that you are definitely not seeing optimal behavior or performance, and the system is thrashing with many messages thinking some member or members are unhealthy.

Potential Resolutions:

The “no longer suspecting” message is really an indication that the member is now considered healthy. However, it also means that the member was considered unhealthy and some member initiated “suspect” processing to determine if we should kick out the member to preserve the integrity and stability of the cluster. You will generally see suspect messages, shown above, for all members, as we send these out across the cluster to gather opinions. Ultimately, if the coordinator finds the member to be unresponsive within member-timeout seconds, the coordinator will kick out the member.

To take action, check the “Reason” seen in some of the logs, and take action accordingly. If this is rare, it is likely not an issue. If frequent, however, you definitely want to research and tune the system to eliminate these messages. If you are seeing the “no longer suspecting” message, that means that you should also see the “Suspecting member” message shown above. However, depending on your version of Tanzu GemFire, it may require debug level logging to see that message.

Seconds Have Elapsed

Log Message:

```
[warn 2021/04/11 02:03:53.220 EDT <ServerConnection on port 10230 Thread 120>
tid=0xac97] 20 seconds have elapsed while waiting for replies:
<PRFunctionStreamingResultCollector 29058 waiting for 1 replies from [XXX]> on
YYY<v18>:10104 whose current membership list is: [LIST OF ALL MEMBERS]
[warn 2021/03/16 02:35:18.588 EDT <Timer-0> tid=0x2e] 15 seconds have elapsed sending a message to XXX(78779)<v4>:565
62 for thread ServerConnection on port 20004 Thread 3
[warn 2021/03/16 02:35:18.588 EDT <Timer-0> tid=0x2e] 15 seconds have elapsed waiting for a direct reply response from XXX
(78779)<v4>:56562 for thread ServerConnection on port 20004 Thread 3
[warn 2021/04/15 03:30:59.107 EDT <main> tid=0x1] 15 seconds have elapsed while
waiting for replies: <DLockRequestProcessor 115 waiting for 1 replies from
[XXX(8582)-server2:68148)<v2>:41000]> on YYY<v2>:41001 whose current membership list
is: [LIST OF ALL MEMBERS]
```

Log Level: warning, fatal. Depending on the operation, the message may be repeated at fatal level if the ack-severe-alert-threshold property has been set and that threshold is also exceeded.

Category: Operations

Meaning:

This message requires action. It is not urgent, but it is an indication that the messaging is taking much longer than expected between peers in your environment. For warn-level logs, the number of seconds displayed maps to the ack-wait-threshold in your environment, which defaults to 15 seconds. For fatal-level logs, the number of seconds displayed maps to the sum of the ack-wait-threshold and the ack-severe-alert-threshold in your environment. ack-severe-alert-threshold is not set by default, meaning that the fatal-level logging will not be seen. Some customers increase this setting, but it is recommended that you understand your environment first and only increase it if deemed necessary after attempting to correct any underlying causes of delays in communications.

Potential Resolutions:

This could be driven by GC related delays, JVM Pauses, burst of activity causing high peer to peer activity, threading issues, overwhelming CPU, etc. You could check for high replyWaitsInProgress across all nodes using JMX or stats analysis. If this is rare, it is not a likely cause for concern. If you are seeing this while experiencing high latency, it is likely an area to focus on. To analyze such issues, we will need all logs, stats, and GC logs across all members to identify which member or members is driving the slowdown.

NOTE: If many members have these messages, while another member does not appear to be waiting for replies from anybody, it is very likely that member is the source of the issue. After analysis to gather some information, you could try bouncing that member to see if this restores the other members to a healthier state.

Message Loss Detected

Message loss is an extremely rare occurrence caused by network instability. This ensures that lost peer-to-peer messages are detected. Log Message described below is logged when a message loss is detected.

Log Message

```
[error 2024/12/11 11:30:57.606 PST server-1 <Disconnect detection thread for shared ordered P2P sender for 192.168.0.29(server-2:36204)<v2>:57629(uid=6) with socket Socket[addr=/192.168.0.29,port=55265,localport=61561] @780896041> tid=0x65] Message loss detected: 3 lost message(s). The lost messages are:
1. NoOpMessage@3d472550 processorId=1 sender=192.168.0.29(server-1:36203)<v1>:50203
2. NoOpMessage@a2172c51 processorId=3 sender=192.168.0.29(server-1:36203)<v1>:50203
3. NoOpMessage@724d27a5 processorId=6 sender=192.168.0.29(server-1:36203)<v1>:50203
The member that did not receive these messages is 192.168.0.29(server-2:36204)<v2>:57629 and the sender is shared ordered P2P sender for 192.168.0.29(server-2:36204)<v2>:57629(uid=6) with socket Socket[addr=/192.168.0.29,port=55265,localport=61562] @652520369
```

Log Level: error

Category: Operations

Meaning:

This message requires immediate action. Peer-to-peer message loss can occur when a socket used for messaging between two cluster members gets unexpectedly disconnected, resulting in the member that sent the message incorrectly assuming that it was successfully received by the member to which it was sent. Lost messages can result in data/state inconsistency and potential data loss, as well as stuck threads in the member that sent the messages, which will wait forever for replies to the lost messages.

This message is logged when we see permanent loss of messages. The receiver member and the peer-to-peer sender which lost the messages are included to assist with debugging.

Potential Resolutions:

There are multiple possible causes of unexpected socket disconnects leading to message loss, but network configuration and instability are likely to be the main driving factors. Message loss should be an extremely rare occurrence, so if message loss is being seen frequently, then consider checking your network configuration to make sure that it's following the recommended settings for GemFire. If message loss is detected, restarting the member which did not receive the messages and/or the member which detected the lost messages will prevent hung threads and data inconsistency.

Member is not responding to heartbeat requests

Log Message:

```
[info 2021/03/29 06:46:56.304 EDT <Geode Failure Detection thread 162> tid=0x474f0c]
received suspect message from myself for XXX(YYY-server1:15972)<v16>:40000: Member
is not responding to heartbeat requests
[info 2021/02/21 00:38:33.436 GMT <Membership Messenger Non-Blocking> tid=0x31] received
suspect message from YYY(cacheServer33010:16994)<v73>:33100 for
ZZZ(gcacheServer33010:27406)<v74>:33100: Member is not responding to heartbeat
requests
[info 2021/02/21 00:38:33.440 GMT <Geode Failure Detection thread 10> tid=0x32f]
Performing availability check for suspect member
XXX(cacheServer33010:27406)<v74>:33100 reason=Member is not responding to heartbeat
requests
```

Log Level: info

Category: Membership

Meaning:

This message requires immediate action. You are in danger of having a member kicked out of the distributed system due to being unresponsive. If the member continues to be unresponsive, the distributed system will kick out the member, to restore stability for the remaining members.

Potential Resolutions:

This is often related to a suboptimal heap or GC configuration. You could be experiencing JVM pauses that require tuning. If you frequently see these messages without having the member kicked out, you have opportunities to tune and eliminate these messages. Alternatively, you could also increase the member-timeout property, however this is only suggested when you have full understanding of what is driving the member to be unresponsive to the heartbeat requests from the member monitoring it.

This message often corresponds with "suspect" messages, and members getting kicked out of the cluster. Logs, stats, and GC logs are required to troubleshoot in this situation.

Enabled-network-partition-detection is set to false

Log Message:

```
[warning 2021/09/11 08:01:41.089 EDT locatorIS2 <Pooled Message Processor 1>
tid=0x48] Creating persistent region _ConfigurationRegion, but
enable-network-partition-detection is set to false. Running with network partition
detection deactivated can lead to an unrecoverable system in the event of a network
split.
```

Log Level: warning

Category: Membership

Meaning:

This is a warning that you have chosen a configuration that makes you more susceptible to data consistency issues if you experience a network partition, or “split brain”. If you do choose this configuration and experience network issues that create a “split brain” scenario, where your distributed system splits into two separate distributed systems (DS), then it is possible that your data will diverge. Specifically, you could do puts into a region in DS A that do not make it into DS B, while also doing puts into DS B that do not make it into DS A. Tanzu GemFire will be unable to resolve this situation for you as you try to recover the system back into a single, healthy DS.

Potential Resolutions:

The best option is to choose to keep enable-network-partition-detection set to true. Beyond that, any split brain driven data divergence will require your manual intervention to avoid possible data loss.

Statistics sampling thread detected a wakeup delay

Log Message:

```
[warning 2021/02/09 21:37:44.728 EST member-49001 <StatSampler> tid=0x36] Statistics
sampling thread detected a wakeup delay of 40203 ms, indicating a possible resource
issue. Check the GC, memory, and CPU statistics.
```

Log Level: warning

Category: Membership

Meaning:

URGENT action is needed. You are experiencing JVM Pauses, where the JVM is preventing Tanzu GemFire from running at all for the given amount of time. This is only logged when the delay is at least three seconds more than your configured statistic-sample-rate. You are vulnerable to having members kicked out of the distributed system.

Potential Resolutions:

This is almost always caused by GC-related behavior. To diagnose such issues, make sure to enable GC logging in your environment. If you have GC logs, search for “Full GC”, “concurrent mode failure”, “exhausted”, and other similar issues that drive long pauses. If you do open a support ticket for assistance, have Tanzu GemFire logs, stats, and GC logs ready to provide to our support team prior to opening the ticket.

If this issue is urgent and you need immediate resolution without having time to fine tune GC, one possible temporary patch is to increase the member-timeout in the gemfire.properties file. This would make Tanzu GemFire more tolerant of processes being somewhat unresponsive for longer durations.

Redundancy has dropped below configured copies

Log Message:

```
[warning 2021/03/23 09:26:51.641 EDT XXX-server01 <PartitionedRegion Message
Processor20> tid=0x1d66] Redundancy has dropped below 2 configured copies to 1 actual
copies for /RegionXYZ
[info 2021/03/23 09:26:51.798 EDT XXX-server01 <PartitionedRegion Message
Processor20> tid=0x1d66] Configured redundancy of 2 copies has been restored to
/RegionXYZ
```

Log Level: warning

Category: Membership

Meaning:

This message requires immediate action to determine if you are now vulnerable to data loss. This message indicates that you have lost access to one of the two configured copies of your data for that RegionXYZ on member XXX-server01. It is not necessarily urgent if you have redundancy configured and capacity for the remaining members to handle the increased load. The corresponding “has been restored” message, an info-level message also shown above, indicates that you have returned to your healthy environment with redundancy in place for that RegionXYZ from the perspective of this member.

Potential Resolutions:

Investigate the cause of the loss in redundancy if it is not already known. For example, it could have been a planned maintenance that drove the cluster below configured redundancy levels. The settings that generally apply here are the number of copies configured, and then, the recovery-delay and startup-recovery-delay settings, which control whether and when we restore redundancy with the loss of a member of the distributed system and when it is added back in. Our documentation discusses these settings in detail.

Rejected connection

Log Message:

```
[warning 2021/05/10 12:28:31.063 BST gfcache.ivapp1237223.croydon.ms.com.7291
<Handshaker /0:0:0:0:0:0:0:0:7291 Thread 10> tid=0xa29] Rejected connection from XXX
because current connection count of 1,600 is greater than or equal to the configured
max of 1,600
[warn 2021/03/28 02:22:01.667 CDT <Handshaker 0.0.0.0/0.0.0.0:40404 Thread 23>
tid=0x85cf] Rejected connection from Server connection from [client host address=YYY;
client port=43198] because incoming request was rejected by pool possibly due to
thread exhaustion
```

Log Level: warning

Category: Communications

Meaning:

This message requires URGENT action. These messages indicate that you have exhausted resources, likely either due to using an insufficient max-connections setting for the cache server configuration or insufficient resources for the level of connection load on the system. Both of these messages are from the same area of code, trying to handle a new client connection request.

Potential Resolutions:

If you have increased load recently, or are using an old, legacy default value of 800 for max-connections, you may want to increase this setting. Many customers use 2000, or even 5000 for those wishing to avoid having Tanzu GemFire throttle their performance or activity by trying to conserve resources.

That said, if this number of connections is unexpected, you are potentially experiencing issues with connection timeouts, driving retry activity and a thrashing of resources that can cause the number of outstanding client connections and threads to be exhausted. You can observe this by examining Tanzu GemFire statistics using a tool like VSD, or, if using JMX, you can monitor usage with the CacheServeMXBean getClientConnectionCount() method. If you ever see unexpected spikes in this value, but are not seeing other symptoms, such as timeouts, perhaps you simply need to increase the max-connections appropriately.

However, if seeing these messages coincides with symptoms like client side timeouts, it could be due to an insufficient read-timeout in the client side pool configuration, or an insufficient accept queue on the server side. Another setting that warrants investigation is the BridgeServer.HANDSHAKE_POOL_SIZE. The default value of BridgeServer.HANDSHAKE_POOL_SIZE is 50. If this number is insufficient, we recommend increasing this Tanzu GemFire system property to a larger value.

PCC service metrics component failing to connect to locator/server

Log Message:


```
{"timestamp":"1620032802.654847383","source":"service-metrics","message":"service-metrics.executing-metrics-cmd","log_leve
l":2,"data":{"error":"exit
status 1","event":"failed","output":"IOException error! MBeanServerConnection failed
to create.\njava.io.IOException: Failed to retrieve RMIServer stub:
javax.naming.ServiceUnavailableException [Root exception is
java.rmi.ConnectException: Connection refused to host:
461737ba-07ca-4897-9e41-a70ae7f26274.server.services.service-instance-94fbf6cc-4073-4a45-8965-7ea855bcd0ca.bosh;
nested exception is: \n\tjava.net.ConnectException: Connection refused (Connection
refused)]\nException in thread \"main\" java.lang.NullPointerException\n\tat
io.pivotal.cloudcache.metrics.cli.MetricsExtractor.lambda$static$0(MetricsExtractor.java:10)\n\tat
io.pivotal.cloudcache.metrics.cli.JMXPropertiesEmitter.lambda$getMemberMetrics$0(JMXPropertiesEmitter.java:55)\n\tat
java.util.stream.ReferencePipeline$3$1.accept(ReferencePipeline.java:193)\n\tat
java.util.Spliterators$ArraySpliterator.forEachRemaining(Spliterators.java:948)\n\tat
java.util.stream.AbstractPipeline.copyInto(AbstractPipeline.java:482)\n\tat
java.util.stream.AbstractPipeline.wrapAndCopyInto(AbstractPipeline.java:472)\n\tat
java.util.stream.ReduceOps$ReduceOp.evaluateSequential(ReduceOps.java:708)\n\tat
java.util.stream.AbstractPipeline.evaluate(AbstractPipeline.java:234)\n\tat
java.util.stream.ReferencePipeline.collect(ReferencePipeline.java:566)\n\tat
io.pivotal.cloudcache.metrics.cli.JMXPropertiesEmitter.getMemberMetrics(JMXPropertiesEmitter.java:56)\n\tat
io.pivotal.cloudcache.metrics.cli.JMXPropertiesEmitter.main(JMXPropertiesEmitter.java:30)\n"}}
{"timestamp":"1620032842.481263161","source":"service-metrics","message":"service-metrics.executing-metrics-cmd","log_leve
l":1,"data":{"event":"starting"}}
```


Category: Communications

Meaning:

Every virtual machine for locators or servers has its own service-metrics component. The job of this component is to periodically check the health of the Tanzu GemFire server/locator processes running. The way it does that job is by making an RMI call to the JMX manager. When it cannot connect to the locator/server process, it starts logging these errors in its own log.

SSLHandshakeException: is disabled

Log Message:

```
[warn 2021/04/26 15:44:52.418 EDT kbc000100.rw.example.com <P2P message reader@388969b8> tid=0x3a] SSL handshake exception
javax.net.ssl.SSLHandshakeException: <<ssl_version>> is disabled
    at sun.security.ssl.InputRecord.handleUnknownRecord(InputRecord.java:637)
    at sun.security.ssl.InputRecord.read(InputRecord.java:527)
    at sun.security.ssl.EngineInputRecord.read(EngineInputRecord.java:382)
    at sun.security.ssl.SSLEngineImpl.readRecord(SSLEngineImpl.java:951)
    at sun.security.ssl.SSLEngineImpl.readNetRecord(SSLEngineImpl.java:896)
    at sun.security.ssl.SSLEngineImpl.unwrap(SSLEngineImpl.java:766)
    at javax.net.ssl.SSLEngine.unwrap(SSLEngine.java:624)
    at org.apache.geode.internal.net.NioSslEngine.handshake(NioSslEngine.java:148)
    at org.apache.geode.internal.net.SocketCreator.handshakeSSLSocketChannel(SocketCreator.java:840)
    at org.apache.geode.internal.tcp.Connection.createIoFilter(Connection.java:1747)
    at org.apache.geode.internal.tcp.Connection.readMessages(Connection.java:1548)
    at org.apache.geode.internal.tcp.Connection.run(Connection.java:1472)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
    at java.lang.Thread.run(Thread.java:748)
```

Category: Communications

Meaning:

This means the specified SSL/TLS protocol is not compatible with, or configured correctly, on one or more members. A typical workaround is to use “any” as the protocol; however, some customers have strict security requirements that mandate specific versions and ciphers, which will require that all members are configured with compatible (matching) protocols and ciphers and that those protocols/ciphers are supported by the underlying JRE.

Unable To Create New Native Thread

Log Message:

```
java.lang.OutOfMemoryError: unable to create new native thread
```

Log Level: warning

Category: Heap / JVM / GC

Meaning:

The JVM needs various resources to create a new “native” thread, which may not map one-to-one with application threads. These resources are external to the JVM heap and include “native” memory for the stack and, potentially, user processes.

Potential Resolution:

Depending on the resource limit encountered, you may need to increase the maximum number of user processes as configured with ulimit and/or “/etc/security/limits.conf”, or you may not have sufficient system memory. In the latter case, you must either make more system memory available, decrease the amount of stack memory used per thread, or both. If you have excess, unused heap under even heavy load, you may be able to reduce the heap size and leave more memory for “native” usage.

Alternatively, you might be able to decrease the stack size of each thread, by setting the JVM parameter “-xss” to something smaller (the defaults are 320 KB for 32-bit JVMs and 1024 KB for 64-bit JVMs), but this must be done with care as it can cause threads to not have enough stack to properly operate. The last and safest option is to add free memory to the system by either adding memory or reducing other consumers of system memory (e.g. other applications).

Available file descriptors in the system: XX is below the set threshold: 50

Log Message:

```
Available file descriptors in the system: 45 is below the set threshold: 50
```

Log Level: warning

Category: Communications

Meaning:

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. If the available sockets falls below the set threshold (which by default is 50), you will start to see this message in the logs.

Potential Resolution:

There are two limits on the maximum descriptors available to a single application, a soft limit, which can be increased using the ulimit command as a user, and a "hard" limit which will require editing "/etc/security/limits.conf" and logging back in. (There is also an OS level limit that will require a system administrator to tune kernel parameters, however, this limit is typically large and is rarely hit.) It is also possible that the FD's being consumed are being driven by a major increase of connections or threads due to some burst of activity or connections timing out. This can lead to retry activity driving the number of open files to increase. If you increase the soft and hard limits, and continue to observe these messages, you may need to analyze whether you have connections timing out, sufficient TCP accept queue, etc. This can require an increase of the p2p.backlog and net.core.somaxconn settings.

An IOException has occurred due to exhaustion of file descriptors. The cache will be closed.

Log Message:

```
An IOException has occurred due to exhaustion of file descriptors. The cache will be closed.
```

Log Level: error

Category: Communications

Meaning:

If the previous file descriptor warning messages are ignored and sockets continued to be opened, the system will eventually log this error message, disconnect the system and close the cache.

Potential Resolution:

None

CommitConflictException

Log Message:

```
org.apache.geode.cache.CommitConflictException: Concurrent transaction commit
detected The key xxx in region /xxx was being modified by another transaction locally
```

Log Level: warning

Category: Operations

Meaning:

You design transactions such that any get operations are within the transaction. This causes those entries to be part of the transactional state, which is desired such that intersecting transactions can be detected and signal commit conflicts. You must catch the commit conflicts in your code using some like the following:

```
try {
  txmgr.begin();
  // add your codes here
  txmgr.commit();
  }
  catch (CommitConflictException conflict) {
    // entry value changed causing a conflict, so try again
  } finally {
  //add your codes here
  }
}
```

Initialization of Region <_B__RegionName_BucketNumber> Completed

Log Message:

```
[info 2021/03/28 00:41:20.047 EDT <Recovery thread for bucket _B__RegionName_32>
tid=0x164] Region _B__RegionName_32 requesting initial image from
IP(gemfire-server-1:88590)<v19>:10100

[info 2021/03/28 00:41:20.048 EDT <Recovery thread for bucket _B__RegionName_32>
tid=0x164] _B__RegionName_32 is done getting image from
IP(gemfire-server-1:88590)<v19>:10100. isDeltaGII is true

[info 2021/03/28 00:41:20.048 EDT <Recovery thread for bucket
 _B__firm_0_RegionName_32> tid=0x164] Region _B__RegionName_32 initialized persistent
 id: /IP:/pathTo-server-1/cldist created at timestamp 1616906479201 version 0
 diskStoreId DiskStoreid name null with data from
 IP(gemfire-server-1:88590)<v19>:10100.

[info 2021/03/28 00:41:20.048 EDT <Recovery thread for bucket _B__RegionName_32>
tid=0x164] Could not initiate event tracker from GII provider
IP(gemfire-server-1:88590)<v19>:10100

[info 2021/03/28 00:41:20.048 EDT <Recovery thread for bucket _B__RegionName_32>
tid=0x164] Initialization of region _B__RegionName_32 completed
```

Log Level: info

Category: Membership

Meaning:

This set of messages are related to the initialization of Partitioned regions. They indicate where the Tanzu GemFire system is retrieving each bucket from to perform this initialization. In the above example, bucket 32 for region “RegionName” is being retrieved from member gemfire-server-1 as Tanzu GemFire believes this to be the most recent data for that bucket. This is the “requesting initial image” message above. The Initialization of region <> completed message can be useful to determine where each specific bucket, for each specific region, is located across the membership.

Potential Resolution:

There is no resolution, but customers have asked how to determine where each bucket exists across the cluster. Using the above message can be very useful to filter the logs to see exactly where each bucket exists in the cluster, for each region. You could use a command such as the following:

```
egrep -R --include=\*.log 'Initialization of region _B__RegionName_' ~/PathToLogFiles//gflogs/*
```

The above command tells you exactly where each bucket exists for region RegionName. If you use only Initialization of region _B__ instead, this would outputs the buckets across all partitioned regions. This output can then be used to know where each specific bucket exists across the cluster, to serve whatever purpose you deem helpful in monitoring your cluster. For more information about how to identify where buckets are located, see How to monitor PR entry and Bucket details in GemFire.

Unknown pdx Type error

Log Message:

```
Caused by: java.lang.IllegalStateException: Unknown pdx type=X
at com.gemstone.gemfire.internal.InternalDataSerializer.readPdxSerializable(InternalDataSerializer.java:2977)
at com.gemstone.gemfire.internal.InternalDataSerializer.basicReadObject(InternalDataSerializer.java:2794)
at com.gemstone.gemfire.DataSerializer.readObject(DataSerializer.java:3212)
at com.gemstone.gemfire.internal.util.BlobHelper.deserializeBlob(BlobHelper.java:81)
at com.gemstone.gemfire.internal.cache.EntryEventImpl.deserialize(EntryEventImpl.java:1407)
at com.gemstone.gemfire.internal.cache.PreferBytesCachedDeserializable.getDeserializedValue(PreferBytesCachedDeserializa
ble.java:65)
at com.gemstone.gemfire.cache.query.internal.index.DummyQRegion.getValues(DummyQRegion.java:153)
at com.gemstone.gemfire.cache.query.internal.index.DummyQRegion.values(DummyQRegion.java:109)
at com.gemstone.gemfire.cache.query.internal.index.DummyQRegion.iterator(DummyQRegion.java:198)
at com.gemstone.gemfire.cache.query.internal.index.CompactRangeIndex$IMQEvaluator.doNestedIterations(CompactRangeInd
ex.java:1763)
at com.gemstone.gemfire.cache.query.internal.index.CompactRangeIndex$IMQEvaluator.evaluate(CompactRangeIndex.java:16
22)
... 27 more
```

Log Level: Error

Category: Operations

Meaning:

A Portable Data eXchange (PDX) related exception that may occur when restarting a distributed system without also restarting any clients. When using PDX serialization without persistence, the above exception may be seen on a client after bouncing all of the servers of the distributed system without restarting the client. Generally, this message indicates that the PDX metadata on some client is unsynchronized with the servers.

Potential Resolution:

To avoid this issue without persisting PDX types on server members, you must restart your client application when restarting the servers. Alternately, to avoid this issue without restarting your client application, you must enable PDX persistence on servers. By doing this, you are guaranteed that any already defined PDX types will remain available between server restarts. This does not require storing the data from your regions, you can store only PDX metadata, regions data, or both. Below mentioned is an example of how to configure PDX persistence on the server side:

```
<disk-store name="pdxDiskStore">
  <disk-dirs>
    <disk-dir>pdxDiskStore</disk-dir>
  </disk-dirs>
</disk-store>

<pdx read-serialized="true" persistent="true" disk-store-name="pdxDiskStore"/>
```

Error calculating expiration

Log Message:

```
2021-06-02 12:35:26,071 FATAL o.a.g.i.c.LocalRegion [Recovery thread for bucket _B__gdc__eventsLow_50] Error calculating e
xpiration An IOException was thrown while deserializing
org.apache.geode.SerializationException: An IOException was thrown while deserializing
        at org.apache.geode.internal.cache.EntryEventImpl.deserialize(EntryEventImpl.java:2041) ~[gemfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.EntryEventImpl.deserialize(EntryEventImpl.java:2032) ~[gemfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.VMCachedDeserializable.getDeserializedValue(VMCachedDeserializable.java:113) ~[g
emfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.LocalRegion.getDeserialized(LocalRegion.java:1280) ~[gemfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.ExpiryRegionEntry.getValue(ExpiryRegionEntry.java:101) ~[gemfire-core-10.0.0.jar:?]
        at com.ihg.enterprise.gdc.model.CustomExpiryHandler.getExpiry(CustomExpiryHandler.java:19) ~[gdc-gemfire-side-2.26-jar
-with-dependencies.jar:?]
        at org.apache.geode.internal.cache.LocalRegion.createExpiryTask(LocalRegion.java:7774) ~[gemfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.LocalRegion.addExpiryTask(LocalRegion.java:7901) ~[gemfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.LocalRegion.addExpiryTask(LocalRegion.java:7753) ~[gemfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.LocalRegion.lambda$rescheduleEntryExpiryTasks$3(LocalRegion.java:7741) ~[gemfire
-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.ExpiryTask.doWithNowSet(ExpiryTask.java:480) [gemfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.LocalRegion.rescheduleEntryExpiryTasks(LocalRegion.java:7739) [gemfire-core-10.0.0.
jar:?]
        at org.apache.geode.internal.cache.LocalRegion.initialize(LocalRegion.java:2394) [gemfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.DistributedRegion.initialize(DistributedRegion.java:1099) [gemfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.BucketRegion.initialize(BucketRegion.java:259) [gemfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.LocalRegion.createSubregion(LocalRegion.java:983) [gemfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.PartitionedRegionDataStore.createBucketRegion(PartitionedRegionDataStore.java:78
4) [gemfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.PartitionedRegionDataStore.grabFreeBucket(PartitionedRegionDataStore.java:459) [ge
mfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.PartitionedRegionDataStore.grabBucket(PartitionedRegionDataStore.java:2875) [gemfi
re-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.ProxyBucketRegion.recoverFromDisk(ProxyBucketRegion.java:463) [gemfire-core-10.
0.0.jar:?]
        at org.apache.geode.internal.cache.ProxyBucketRegion.recoverFromDiskRecursively(ProxyBucketRegion.java:406) [gemfir
e-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.PRHARedundancyProvider$2.run2(PRHARedundancyProvider.java:1640) [gemfire-cor
e-10.0.0.jar:?]
        at org.apache.geode.internal.cache.partitioned.RecoveryRunnable.run(RecoveryRunnable.java:60) [gemfire-core-10.0.0.ja
r:?]
        at org.apache.geode.internal.cache.PRHARedundancyProvider$2.run(PRHARedundancyProvider.java:1630) [gemfire-core-
10.0.0.jar:?]
        at java.lang.Thread.run(Thread.java:748) [?:1.8.0_212]
Caused by: java.io.IOException: Unknown header byte 83
        at org.apache.geode.internal.serialization.DscodeHelper.toDSCODE(DscodeHelper.java:40) ~[gemfire-serialization-10.0.0.j
ar:?]
        at org.apache.geode.internal.InternalDataSerializer.basicReadObject(InternalDataSerializer.java:2494) ~[gemfire-core-10.0.
0.jar:?]
        at org.apache.geode.DataSerializer.readObject(DataSerializer.java:2864) ~[gemfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.util.BlobHelper.deserializeBlob(BlobHelper.java:99) ~[gemfire-core-10.0.0.jar:?]
        at org.apache.geode.internal.cache.EntryEventImpl.deserialize(EntryEventImpl.java:2039) ~[gemfire-core-10.0.0.jar:?]
        ... 24 more
```

Log Level: Warning

Category: Storage

Potential Resolution:

This is due to inconsistencies in the data stored on region or disk versus the PdxType and may throw during deserialization. Cleaning the data or syncing it according to PdxType is a possible solution.

PdxType limitations for GFSH queries

Log Message:

```
[info 2021/06/15 13:01:24.238 EDT 170834GFCluster.sd-1d7e-bd1c.cacheServer40404 <Function Execution Processor3> tid=0x
7b] Exception occurred:
org.apache.geode.pdx.JSONFormatterException: Could not create JSON document from PdxInstance
	at org.apache.geode.pdx.JSONFormatter.toJSON(JSONFormatter.java:173)
	at org.apache.geode.management.internal.cli.domain.DataCommandResult$SelectResultRow.valueToJson(DataCommand
Result.java:726)
	at org.apache.geode.management.internal.cli.domain.DataCommandResult$SelectResultRow.resolveStructToColumns(Dat
aCommandResult.java:712)
	at org.apache.geode.management.internal.cli.domain.DataCommandResult$SelectResultRow.resolveObjectToColumns(Da
taCommandResult.java:689)
	at org.apache.geode.management.internal.cli.domain.DataCommandResult$SelectResultRow.createColumnValues(DataC
ommandResult.java:679)
	at org.apache.geode.management.internal.cli.domain.DataCommandResult$SelectResultRow.<init>(DataCommandResult.j
ava:662)
	at org.apache.geode.management.internal.cli.functions.DataCommandFunction.createSelectResultRow(DataCommandFun
ction.java:266)
	at org.apache.geode.management.internal.cli.functions.DataCommandFunction.select_SelectResults(DataCommandFuncti
on.java:252)
	at org.apache.geode.management.internal.cli.functions.DataCommandFunction.select(DataCommandFunction.java:220)
	at org.apache.geode.management.internal.cli.functions.DataCommandFunction.select(DataCommandFunction.java:173)
	at org.apache.geode.management.internal.cli.functions.DataCommandFunction.execute(DataCommandFunction.java:122)
	at org.apache.geode.internal.cache.MemberFunctionStreamingMessage.process(MemberFunctionStreamingMessage.java:
193)
	at org.apache.geode.distributed.internal.DistributionMessage.scheduleAction(DistributionMessage.java:367)
	at org.apache.geode.distributed.internal.DistributionMessage$1.run(DistributionMessage.java:430)
	at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
	at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java)
	at org.apache.geode.distributed.internal.ClusterDistributionManager.runUntilShutdown(ClusterDistributionManager.java:95
2)
	at org.apache.geode.distributed.internal.ClusterDistributionManager.doFunctionExecutionThread(ClusterDistributionManag
er.java:806)
	at org.apache.geode.internal.logging.LoggingThreadFactory.lambda$newThread$0(LoggingThreadFactory.java:121)
	at java.lang.Thread.run(Thread.java:748)
Caused by: java.lang.IllegalStateException: PdxInstance returns unknwon pdxfield value for type Wed Apr 07 00:00:00 EDT 202
1
	at org.apache.geode.pdx.internal.json.PdxToJSON.writeValue(PdxToJSON.java:144)
	at org.apache.geode.pdx.internal.json.PdxToJSON.getJSONString(PdxToJSON.java:175)
	at org.apache.geode.pdx.internal.json.PdxToJSON.writeValue(PdxToJSON.java:138)
	at org.apache.geode.pdx.internal.json.PdxToJSON.getJSONString(PdxToJSON.java:175)
	at org.apache.geode.pdx.internal.json.PdxToJSON.writeValue(PdxToJSON.java:138)
	at org.apache.geode.pdx.internal.json.PdxToJSON.getJSONString(PdxToJSON.java:175)
	at org.apache.geode.pdx.internal.json.PdxToJSON.writeValue(PdxToJSON.java:138)
	at org.apache.geode.pdx.internal.json.PdxToJSON.getJSONString(PdxToJSON.java:175)
	at org.apache.geode.pdx.internal.json.PdxToJSON.writeValue(PdxToJSON.java:138)
	at org.apache.geode.pdx.internal.json.PdxToJSON.getJSONString(PdxToJSON.java:175)
	at org.apache.geode.pdx.internal.json.PdxToJSON.getJSON(PdxToJSON.java:57)
	at org.apache.geode.pdx.JSONFormatter.toJSON(JSONFormatter.java:171)
	... 19 more
```

Log Level: INFO

Category: Operations

Potential Resolution:

Other than primitive types like object types (String, Character, Date etc.) will not be deserialized on GFSH queries.

Apache.Geode.Client.AllConnectionsInUseException

Log Message:

In StdOut/StdError on Client Side:

```
Apache.Geode.Client.AllConnectionsInUseException
Region::getAll: All connections are in use
Apache.Geode.Client.Region`2[[System.__Canon, mscorlib],[System.__Canon, mscorlib]].GetAll(System.Collections.Generic.ICollection`1<System.__Canon>, System.Collections.Generic.IDictionary`2<System.__Canon,System.__Canon>, System.Collections.Generic.IDictionary`2<System.__Canon,System.Exception>, Boolean)
   Apache.Geode.Client.Region`2[[System.__Canon, mscorlib],[System.__Canon, mscorlib]].GetAll(System.Collections.Generic.ICollection`1<System.__Canon>, System.Collections.Generic.IDictionary`2<System.__Canon,System.__Canon>, System.Collections.Generic.IDictionary`2<System.__Canon,System.Exception>)
```

Category: Operations

Meaning:

This is evidence of the connection pool getting overwhelmed on the client side and not a problem on the Tanzu GemFire server side.

Resolution: Increase the max-connections property to higher value as appropriate on pool settings on native client.

org.apache.geode.pdx.PdxInitializationException

Log Message: / Stack-trace / StdError:

```
The Cache Server process terminated unexpectedly with exit status 1. Please refer to the log file in /appdata/gemfire/edl/data/server for full details.
Exception in thread "main" org.apache.geode.pdx.PdxInitializationException: Could not create pdx registry
   at org.apache.geode.pdx.internal.PeerTypeRegistration.initialize(PeerTypeRegistration.java:204)
   at org.apache.geode.pdx.internal.TypeRegistry.creatingDiskStore(TypeRegistry.java:267)
   at org.apache.geode.internal.cache.DiskStoreFactoryImpl.create(DiskStoreFactoryImpl.java:160)
   at org.apache.geode.internal.cache.xmlcache.CacheCreation.createDiskStore(CacheCreation.java:792)
   at org.apache.geode.internal.cache.xmlcache.CacheCreation.initializePdxDiskStore(CacheCreation.java:783)
   at org.apache.geode.internal.cache.xmlcache.CacheCreation.create(CacheCreation.java:507)
   at org.apache.geode.internal.cache.xmlcache.CacheXmlParser.create(CacheXmlParser.java:338)
   at org.apache.geode.internal.cache.GemFireCacheImpl.loadCacheXml(GemFireCacheImpl.java:4294)
```

Category: Operations

Meaning:

This explains that the PdxRegion is not getting initialized due to the corrupted cluster configs.

Potential Resolution:

Stop locators, then clear the cluster configs and pdx disk stores, then start the locators. For more information, see Fails to Start a Cache Member with “org.apache.geode.pdx.PdxInitializationException: Could not create pdx registry” .

Format of the string «cache xml file's content» used for parameterization is unresolvable

Note: The spelling "perameterization" is incorrect in the codebase.

Log Message:

```
[error 2021/09/08 11:42:16.830 EDT <main> tid=0x1] Format of the string <?xml version="1.0"?>
<cache xmlns="http://geode.apache.org/schema/cache"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cache-1.0.xsd"
    version="1.0">
<gateway-sender id="GW_SENDER_${REMOTE_DS_ID_1}" remote-distributed-system-id="${REMOTE_DS_ID_1}" manual-sta
rt="${GW_START_ID_1}" parallel="false" enable-persistence="true" disk-store-name="gw1" disk-synchronous="true" dispatcher-
threads="5" socket-read-timeout="300000"/>
<gateway-sender id="GW_SENDER_${REMOTE_DS_ID_2}" remote-distributed-system-id="${REMOTE_DS_ID_2}" manual-sta
rt="${GW_START_ID_2}" parallel="false" enable-persistence="true" disk-store-name="gw2" disk-synchronous="true" dispatcher-
threads="5" socket-read-timeout="300000"/>
  <gateway-receiver hostname-for-senders="${HOSTNAME_FOR_SENDERS}" bind-address="${HOSTNAME}" start-port="153
1" end-port="1532"/>
-
-
-
</cache>
 used for parameterization is unresolvable


[error 2021/09/08 11:42:17.260 EDT <main> tid=0x1] Cache initialization for GemFireCache[id = -29791992; isClosing = false; is
ShutDownAll = false; created = Wed Sep 08 11:39:37 EDT 2021; server = false; copyOnRead = false; lockLease = 120; lockTime
out = 60] failed because: java.lang.NullPointerException
```

Category: Configuration

Meaning:

This error occurs when the parameterized values are provided for the properties/attributes. For example, manual-start="${GW_START_ID_1}". This property expects boolean, but seems to be getting some non-boolean value.

Potential Resolution:

Fix the errors for incorrect values or their types when provided dynamically.

RegionExistException

Log Message:

```
[error 2021/09/29 11:25:48.885 CDT <main> tid=0x1] Cache initialization for
GemFireCache[id = 755944228; isClosing = false; isShutDownAll = false; created = Wed
Sep 29 11:25:46 CDT 2021; server = false; copyOnRead = false; lockLease = 120;
lockTimeout = 60] failed because: org.apache.geode.cache.CacheXmlException: While
parsing XML, caused by org.xml.sax.SAX
Exception: A CacheException was thrown while parsing XML.
org.apache.geode.cache.RegionExistsException: /RegionX
```

Log Level: Error

Category: Configuration

Meaning:

This message indicates that the locator already has region (RegionX) in the cluster configuration and while starting up, a duplicate region is being created via API or cache.xml.

Potential Resolutions:

Remove duplicate region definition from the configurations.

- If enable-cluster-configuration=true exists in locator properties, then do the following:
    - Export the cluster configuration (export cluster-configuration --xml-file=value)
    - Remove the duplicate Region definition
    - Re-import the cluster configuration (import cluster-configuration --action=STAGE) and restart.
- If enable-cluster-configuration=false exists in locator properties, then remove the duplicate region definition from cache.xml.

Missing Diskstore Exception

Log Message:

```
Region /RegionX 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: /192.0.2.0:/home/grawat/server1/.

Members with potentially new data:
[
  DiskStore ID: 160d415538c44ab0-9f7d97bae0a2f8de
  Name: server2
  Location: /192.0.2.0:/home/grawat/server2/.
]
Use the "gfsh show missing-disk-stores" command to see all disk stores
that are being waited on by other members.
```

Log Level: Info

Category: Storage

Meaning:

When you start a member with a persistent region, the data is retrieved from disk stores to recreate the member's persistent region. If the member does not hold all of the most recent data for the region, then other members have the data, and region creation blocks, waiting for those other members. A partitioned region with colocated entries also blocks on start up, waiting for the entries of the colocated region to be available. This message shows that the disk store for server2 has the most recent data for the region, and server1 is waiting for server2.

Potential Resolutions:

- Start all members with persisted data first and at the same time.
- Respond to the waiting members by starting the server on which the waiting member is waiting on.

Could not create an instance of a class

Log Message:

```
Could not create an instance of a class com.xxx.yyy.zzz
```

Log Level: Error

Category: Configuration

Meaning:

This message indicates that either the class is not available in the classpath or the jar, which contains this class, is not deployed on Cache servers.

Potential Resolutions:

- Make the class available to classpath.
- Deploy the class on cache servers.

PartitionedRegion#cleanupFailedInitialization: Failed to clean the PartitionRegion allPartitionedRegions

Log Message:

```
[warning 2021/05/15 08:51:46.460 EDT 170834GFCluster.lmwcpbacap01p.cacheServer50506
<main> tid=0x1] PartitionedRegion#cleanupFailedInitialization: Failed to clean the
PartionRegion allPartitionedRegions
org.apache.geode.distributed.DistributedSystemDisconnectedException: Distribution
manager on 10.102.8.41(cacheServer50506:278621)<v1>:50001 started at Sat May 15
08:44:31 EDT 2021: Failed to acknowledge a new membership view and then failed tcp/ip
connection attempt, caused by org.apache.geode.ForcedDisconnectException: Failed to
acknowledge a new membership view and then failed tcp/ip connection attempt
```

Log Level: Error

Category: Membership

Meaning:

This message indicates that the buckets for partitioned regions have not recovered fully but a destroy region is issued for the regions whose buckets are still recovering.

Potential Resolutions:

Make sure that regions are recovered before issuing any destroy command.

Could not find any server to create primary client queue on.

Log Message:

```
[error 2016/09/13 10:45:29.351 PDT client tid=0x34] Could not find any server to
create primary client queue on. Number of excluded servers is 0 and the exception is
null.
```

Log Level: Error

Category: Communications

Meaning:

When a client with subscription-enabled="true" is started, messages like the one below will be logged in the Tanzu GemFire client log. If subscription-redundancy is not set, there will be one of these; if it is set to 1, there will be two, etc. The Cache Client Updater Thread is the thread waiting for events from the server. If

no other server is available to which the Cache Client Updater Thread is connected, then above error message will be logged:

Potential Resolutions:

Make sure that the server, to which the Cache Client Updater Thread is connected, is up and running.

Cluster configuration service not available

Log Message:

```
Exception in thread "main" org.apache.geode.GemFireConfigException: cluster configuration service not available
  at org.apache.geode.internal.cache.GemFireCacheImpl.requestSharedConfiguration(GemFireCacheImpl.java:1265)
```

Log Level: Error

Category: Configuration

Meaning:

This message indicates that the cache server is configured with, “use-cluster-configuration = true”, but is unable to get the cluster configuration from the locator.

Potential Resolutions:

Ensure that the locator has “enable-cluster-configuration=true” and the cache servers can get the cluster configurations from locators.

The secondary map already contained an event from hub null so ignoring new event

Log Message:

```
[warn 2021/09/10 15:49:31.692 CEST <P2P message reader for
00.01.02.03(some-node:17718)<v2>:41001 unshared ordered uid=1086 dom #1 port=39166>
tid=0x247bb] AsyncEventQueue_SubstitutionEventQ: The secondary map already contained
an event from hub null so ignoring new event GatewaySenderEventImpl[id=EventID[id=25
bytes...
```

Log Level: Warn

Category: Operations

Meaning:

This message indicates that the secondary gateway sender, hosted by a server, received an event that was already processed by the primary gateway sender of other server B; so the event itself should not be added to the internal map of unprocessed events.

Potential Resolutions:

This message, if seen occasionally, is harmless in most situations.

Create is present in more than one Oplog. This should not be possible. The Oplog Key ID for this entry is

Log Message:

```
java.lang.AssertionError: Oplog::readNewEntry: Create is present in more than one Oplog. This should not be possible
```

Log Level: Error

Category: Storage

Meaning:

This error indicates that the oplog is corrupted which makes it harder to write/delete any new entry in the oplogs (Diskstores) because of which cache servers will have trouble starting.

Potential Resolutions:

Clean up the disk stores.

Detected conflicting PDX types during import

Log Message:

```
Could not process command due to error. Detected conflicting PDX types during import
```

Log Level: Error

Category: Operations

Meaning:

When data is imported into a cluster with pdx serialization with existing data using gfsh import/export command and if the receiving cluster already has some data with different pdx metadata, the import will fail with the error.

Potential Resolutions:

Import data in the empty cluster or programmatically read the .gfd file and then perform the put operation.

A tenured heap garbage collection has occurred

Log Message:

```
[info 2021/10/13 17:14:56.177 EDT memberXXX <Notification Handler1> tid=0x69] A
tenured heap garbage collection has occurred.  New tenured heap consumption:
492250720
```

Log Level: Info

Category: Heap/GC/JVM/OS

Meaning:

This message occurs when a tenured space garbage collection has occurred. The goal is to provide the customer with a very accurate read for how much heap is actually consumed. External monitors do not know when a collection has occurred. The value specified is how much live data exists in tenured heap.

If you see this value constantly increasing over time, without a similar rate of increase of Tanzu GemFire entries, then this warrants some investigation into potential leaks. Short term increases due to queries, for example, are not worthy of concern, other than providing an indication that finer tuning may be warranted. The short term data resulting from a query would hopefully be fulfilled using the young generation heap, most of the time.

Potential Resolutions:

No resolution necessary. This is informative only. If you see this message frequently, however, it is a sign that you may need more heap, or finer tuning. You may be imbalanced unknowingly, etc. If seeing this message more frequently than every 1 hour, consistently, it is a sign that you may need tuning. Note:

G1GC “mixed” collections may not drive this message, unless you are using more current versions of the JDK.

Allocating larger network read buffer

Log Message:

```
[info 2021/10/13 17:14:56.177 EDT locator <P2P message reader for
192.168.1.5(server1:8438)<v1>:41001 shared unordered sender uid=1 local port=42181
remote port=57345> tid=0x4c] Allocating larger network read buffer, new size is
134074 old size was 32768.
```

Log Level: Info

Category: Communications

Meaning:

This may require configuration change, to give more optimal behavior. If you have different socket-buffer-sizes across the various members of your distributed system, including locators, this message may be a sign that messages are potentially being lost. This can lead to distributed deadlocks. This message essentially means that the system is needing to grow and shrink to handle the messaging between the members.

Potential Resolutions:

Set all members, including locators, to the same socket-buffer-size. If you have seen this message, and appear to be impacted in the system, it may warrant some deeper analysis of the health of the system. Check for stuck threads, potentially gather thread dumps, to assess whether you are impacted.

Socket send buffer size is instead of the requested

Log Message:

```
[info 2021/11/19 13:51:47.569 PST server1 <P2P message reader@75099de0> tid=0x30]
Socket send buffer size is 6710884 instead of the requested 16777215.
```

Log Level: Info

Category: Communications

Meaning:

This may require configuration change, to give more optimal behavior. This message tells you that your Tanzu GemFire configuration is specifying a larger socket-buffer-size that the lower OS is going to permit. Hence, you see this message, and perhaps less than optimal behavior.

Potential Resolutions:

Make sure to set all members OS configurations to be the same, similar enough to avoid having this less than optimal potential chunking of messages when sending messages between members of the Tanzu GemFire distributed system.

quorum has been lost

Log Message:

```
[warn 2021/12/03 23:02:41.026 EST <Geode Membership View Creator> tid=0x347] total
weight lost in this view change is 65 of 111.  Quorum has been lost!
```

Log Level: warn

Category: Membership

Meaning: This message requires URGENT attention. It is closely associated with other messages, but indicates that the membership is very unhealthy, and you have potentially lost your entire cluster, or are having some “split brain” behavior, etc.

The above example message shows that a total weight of 65 has been lost, out of 111. This is greater than 50% of the weight, in one view change, hence driving the loss of quorum. When this much weight has been lost, it is generally something affecting the network connectivity, versus a GC event. Please read our extensive documentation on member weight, network partitions, etc.

Potential Resolutions:

It depends mostly on how many members have been removed, and it is possible that the entire cluster has gone down as a result of this loss of quorum. If you have enable-network-partition-detection=true, as we recommend, it is possible to lose the entire cluster if you see the above message. If most of the membership weight has crashed, for example, the losing side does not know that, but the losing side (i.e. the side with less weight) will shut itself down, even though it includes the only still running members. Restart members to restore your cluster to full health, and determine the root cause for why so many members crashed simultaneously.

possible loss of quorum due to the loss of cache processes

Log Message:

```
[fatal 2021/12/03 23:02:41.027 EST <Geode Membership View Creator> tid=0x347]
Possible loss of quorum due to the loss of 6 cache processes: [<list of the ips and
processes>]
```

Log Level: fatal

Category: Membership

Meaning: This is very closely tied to the “quorum has been lost” message. They will often go hand in hand, and potentially even out of order, where you will see the “possible loss” after the “has been lost” message.

Potential Resolutions:

Follow the guidance provided in the, “quorum has been lost,” message. We definitely recommend having enable-network-partition-detection=true set to protect you from split brain driving the data in your split (now 2) distributed systems from diverging and becoming unrecoverable without manual intervention.

Membership service failure: Exiting due to possible network partition event due to loss of <n> cache processes

Log Message:

```
[fatal 2021/12/03 23:02:42.028 EST <Geode Membership View Creator> tid=0x347]
Membership service failure: Exiting due to possible network partition event due to
loss of 6 cache processes: [<list of the 6 cache processes lost, in this example>]
```

Note: This message generally comes with a full stack trace showing the forceDisconnect.

Log Level: fatal

Category: Membership

Meaning: This message requires URGENT attention. It is closely associated with other loss of quorum messages, but indicates that the membership is very unhealthy, and you have potentially lost your entire cluster, or are having some “split brain” behavior, etc.

Potential Resolutions:

Follow the guidance provided in the, “quorum has been lost,” message. VMware recommends that you set enable-network-partition-detection=true to protect you from split brain driving the data in your split (now 2) distributed systems diverging and becoming unrecoverable without manual intervention. VMware also recommends that you investigate to determine whether some network event drove the Tanzu GemFire cluster into this state due to an inability to communicate across the distributed system.

had a weight of

Log Message:

```
[info 2021/12/09 23:19:55.100 EST memberXXX <Geode Membership View Creator> tid=0x57]
memberXXX)<v36>:50001 had a weight of 10
```

Log Level: info

Category: Membership

Meaning: This message indicates that a member has either crashed, or has been kicked out of the distributed system. By default, locators have a weight of 3, LEAD cache server has a weight of 15, and other cache servers have a weight of 10. In the example message, given the weight of 10, you would know that the member that has been kicked out is a non-lead cache server. Depending on your topology, and the number of members in your distributed system, the loss of one such cache server may not impact you much at all.

Potential Resolutions:

You should try to determine the cause of the member leaving the distributed system. If you have auto-reconnect enabled, as you would by default, the member may rejoin automatically, unless it is a crash. If the member was kicked out due to being unresponsive, it may have auto-reconnected, restoring you to a fully running cluster. That said, you likely need to run a rebalance to evenly distribute data, or primary buckets if using partitioned regions. You may require GC tuning, etc. If you do not understand the cause of the membership change, you should reach out for assistance.

An additional Function Execution Processor thread is being launched

Log Message:

```
[warn 2021/12/01 21:29:56.689 EST memberXXX2 <Function Execution Processor1>
tid=0x27] An additional Function Execution Processor thread is being launched because
all <n> thread pool threads are in use for greater than <t> ms
```

Log Level: warn

Category: Operations

Meaning: This requires some action to achieve optimal behavior. If you see this message, it means that your normal behavior requires more than the configured number of function execution threads, set using

DistributionManager.MAX_FE_THREADS. The default has increased recently, but if you see this message, regardless of the current setting <n> shown in the example message, it indicates that your function executions will potentially take longer, due to Tanzu GemFire behavior.

Potential Resolutions:

If you see this message, then you should increase your DistributionManager.MAX_FE_THREADS configured setting, until you have eliminated such messages. You may want to consider the same for your DistributionManager.MAX_THREADS and DistributionManager.MAX_PR_THREADS settings, if not recently updated based on your current operations and load in the system.

Sending new view

Received new view

Admitting member

Log Message:

```
[info 2021/11/20 00:05:31.381 EST gbe-louweblps175(8551)-locator1 <Geode Membership
View Creator> tid=0xd34] sending new view View[<coordinator member
info>:48414:locator)<ec><v314>:41000|342] members: [<current member list>] shutdown:
[<member that shut down>:42306)<v315>:41003]
[info 2021/09/29 01:41:30.472 EDT <memberXXX> tid=0x1d] received new view:
View[<coordinator member>)<ec><v0>:50000|5] members: [list of members, indicating
shutdown, crashed members] old view is: <previous view information, including list of
members and state>

[info 2021/12/01 21:36:21.966 EST DCS-DCS-CLUSTER-10.104.39.130-dmnode-002 <View
Message Processor1> tid=0x3c] Admitting member <<memberXXX:26842)<v6>:10131>. Now
there are 6 non-admin members.
```

Log Level: info

Category: Membership

Meaning: These messages can be very helpful to understand who the coordinator of the Distributed System is, the lead cache server member, and the change in state of the membership, whether members are leaving or joining the distributed system. This will include the cause of leaving, whether a “graceful” shutdown, or a “crash”. You will only ever see the “Sending new view” message in the current coordinator of the system at that time. All members receive this view, and admit the member to the membership list. You only have a full membership when you see the expected number of non-admin members, six in the above, “Now there are 6 non-admin members,” example.

Potential Resolutions:

These are informational only, but if you do see unexpected membership changes, which drive these “new view” messages, you can search the logs for these messages to see whether it was considered graceful, a crash, etc., and look for other logging messages which likely provide additional insight.

Member at unexpectedly left the distributed cache

Log Message:

```
[info 2022/01/11 04:35:34.242 EST <View Message Processor1> tid=0x89] Member at
<memberXXX>:3853)<v11>:10104 unexpectedly left the distributed cache: departed
membership view
```

Log Level: info

Category: Membership

Meaning: This message is an indication that a member has experienced a non-graceful removal from the distributed system. This will then correspond with “new view” messages being sent to all members of the DS, showing the member in the list of “crashed” members.

Potential Resolutions:

This specific message does not indicate much other than the change in ownership. Search for other messages across the cluster which may indicate the reason, such as being unresponsive. Perhaps it is due to not responding to “heartbeat” messages. With auto reconnect, it is possible that the membership has been restored to a full membership, but it is also important to check on the balance of data and load. A rebalance may be prudent to restore the balance in the system. This includes redistributing primary buckets for partitioned regions, which is generally a good idea after any changes in ownership, when time permits.

Cache server: failed accepting client connection

Remote host closed connection during handshake

SSL peer shut down incorrectly

Log Message:

```
[warn 2021/12/01 21:26:11.216 memberXXX <Handshaker /10.104.39.130:10102 Thread 8831> tid=0x2d19e] Cache server: fail
ed accepting client connection javax.net.ssl.SSLHandshakeException: Remote host closed connection during handshake
javax.net.ssl.SSLHandshakeException: Remote host closed connection during handshake
    at sun.security.ssl.SSLSocketImpl.readRecord(SSLSocketImpl.java:994)
    at sun.security.ssl.SSLSocketImpl.performInitialHandshake(SSLSocketImpl.java:1367)
    at sun.security.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:1395)
    at sun.security.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:1379)
    at org.apache.geode.internal.net.SocketCreator.handshakeIfSocketIsSSL(SocketCreator.java:1094)
    at org.apache.geode.internal.cache.tier.sockets.AcceptorImpl.getCommunicationModeForNonSelector(AcceptorImpl.java:1
559)
    at org.apache.geode.internal.cache.tier.sockets.AcceptorImpl.handleNewClientConnection(AcceptorImpl.java:1431)
    at org.apache.geode.internal.cache.tier.sockets.AcceptorImpl.lambda$handOffNewClientConnection$4(AcceptorImpl.java:1
342)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
    at java.lang.Thread.run(Thread.java:748)
Caused by: java.io.EOFException: SSL peer shut down incorrectly
    at sun.security.ssl.InputRecord.read(InputRecord.java:505)
    at sun.security.ssl.SSLSocketImpl.readRecord(SSLSocketImpl.java:975)
```

Log Level: warn

Category: Membership

Meaning: While this looks to be very SSL/TLS specific, this message is often driven by the many of the same client connectivity issues as in the non-SSL/TLS case. This is a client-server connection that is failing because the connection terminated. Besides the general client-server connectivity issues, however, this could also be caused when the client cannot validate the server’s Certificate, and so hangs up. This

message does not indicate any reasons for why that connectivity was lost, but does indicate client-server connectivity issues and the cause needs to be investigated and understood.

Potential Resolutions:

Review client logs to see if there's anything informative there, such as SSL/TLS validation issues, and then investigate logs and stats for possible connectivity or performance issues on the server.

Function: cannot be executed because the members [list of members] are running low on memory

Log Message:

```
[error 2022/01/11 03:49:41.307 EST <ServerConnection on port 10230 Thread 5>
tid=0x28d] <function info> Function: <functionName> cannot be executed because the
members [<list of members>)<v3>:10104] are running low on memory
```

Log Level: error

Category: Operations, Storage

Meaning: This is very similar to the "canceled" query message, but applies to function executions. Essentially, before execution the system recognizes the heap has surpassed the critical-threshold in some subset of members, and therefore the system chooses not to begin the function execution. You should also see the "above heap critical threshold" message in some logs if seeing this message.

Potential Resolutions:

Please follow the same guidelines as the "Query execution canceled due to memory threshold" message.

Region bucket has persistent data that is no longer online stored at these locations

Log Message:

```
[error 2022/01/11 03:51:41.809 EST <ServerConnection on port 10230 Thread 2>
tid=0x21f] <filtered>:Region <regionName> bucket 51 has persistent data that is no
longer online stored at these locations: [<list of members hosting the bucket
including timestamp information>l]
```

Log Level: error

Category: Membership

Meaning: This message tells us that we have lost access to some persistent copy of the given bucket ("51" in the above example). So we know we have a partitioned persistent region where some of the hosting members are not available.

Potential Resolutions:

Determine the cause of the loss of the given member or members hosting that bucket, provided in the message. We do not recommend executing any gfsh "revoke" command without expert interaction and assistance. It is possible you could cause a loss of data.

Region <regionName> has potentially stale data. Buckets [list] are waiting for another offline member

Log Message:

```
[info 2021/12/03 06:52:17.226 EST <PersistentBucketRecoverer for region <r>
tid=0x147] Region <r> (and any colocated sub-regions) has potentially stale data.
Buckets [27, 85, 92] are waiting for another offline member to recover the latest
data.My persistent id is:
  DiskStore ID: <disk store id>
  Name: <member name>
  Location: <member location>
Offline members with potentially new data:[
  DiskStore ID: <disk store id of member with potentially newer data>
  Location: <member location>
  Buckets: [27, 85, 92]
] Use the gfsh show missing-disk-stores command to see all disk stores that are being waited on by other members.
```

Log Level: info

Category: Membership

Meaning: This message normally shows when a member is starting, during bucket recovery of partitioned regions, and indicates that it is waiting for other members, where the data is considered to be more current, to start. Once those members start, the latest copy of the data will be accessible and the member will perform a GII (get initial image) to recover the buckets allowing the member to proceed with initialization.

Potential Resolutions:

If this message appears, perhaps you have members not yet started, and you need to start those members. We recommend starting all cache server processes simultaneously, especially after a clean shutdown, so that the subsequent startup has access to all buckets, and no member is stuck waiting for other members to start.

If you did not have a clean shutdown, or some member has been down for a long time, do NOT start up that member as the first member of a cluster. Otherwise, you will get into a ConflictingDatePersistenceException state that will then require revoking disk stores.

This is a completely avoidable scenario. It is better to start all of the members that have been up and part of the healthy cluster first, and then add back that member later, to be able to get that member up to date, with the latest copies of the buckets loaded from other members. If you see this message, you may want to check current status with the gfsh “show metrics” command to determine whether your number of buckets without redundancy is changing for the specified region over time. If not, you should definitely take a thread dump across all members to determine whether you are having some form of distributed deadlock issue during startup. It is possible that you are simply having major contention/congestion due to some insufficient configuration, such as DistributionManager.MAX_PR_THREAD or DistributionManager.MAX_THREADS. This can be evaluated by analyzing the statistics of the system using a tool like VSD.

Cluster membership failed due to ForcedDisconnectException

Log Message:

```
[error 2025/09/30 07:44:41.829 PDT locator1 <DisconnectThread> tid=0x8d] cluster membership failed due to
org.apache.geode.ForcedDisconnectException: Exiting due to possible network partition event due to loss of 1 cache processes:
[192.168.37.245(server1:190)<v1>:50571]. No thread delays observed on this member. System metrics in the last 59 seconds
{CPU usage=33.1%, Load Average=30.18, Membership thread delay=0 seconds, TCP segment retransmits=0%, GC collection ti
me=0 ms}
    at gemfire//org.apache.geode.distributed.internal.ClusterDistributionManager$DMListener.membershipFailure(ClusterDistrib
utionManager.java:2156)
    at gemfire//org.apache.geode.distributed.internal.membership.gms.GMSMembership.uncleanShutdown(GMSMembership.j
ava:1177)
    at gemfire//org.apache.geode.distributed.internal.membership.gms.GMSMembership$ManagerImpl.lambda$uncleanShutdo
wnDS$0(GMSMembership.java:1763)
    at java.base/java.lang.Thread.run(Thread.java:829)
Caused by: org.apache.geode.distributed.internal.membership.api.MemberDisconnectedException: Exiting due to possible netw
ork partition event due to loss of 1 cache processes: [192.168.37.245(server1:190)<v1>:50571]
    at gemfire//org.apache.geode.distributed.internal.membership.gms.GMSMembership$ManagerImpl.forceDisconnect(GMS
Membership.java:1774)
    at gemfire//org.apache.geode.distributed.internal.membership.gms.membership.GMSJoinLeave.forceDisconnect(GMSJoin
Leave.java:1132)
    at gemfire//org.apache.geode.distributed.internal.membership.gms.membership.GMSJoinLeave.access$1300(GMSJoinLea
ve.java:89)
    at gemfire//org.apache.geode.distributed.internal.membership.gms.membership.GMSJoinLeave$ViewCreator.prepareAndS
endView(GMSJoinLeave.java:2573)
    at gemfire//org.apache.geode.distributed.internal.membership.gms.membership.GMSJoinLeave$ViewCreator.createAndSe
ndView(GMSJoinLeave.java:2550)
    at gemfire//org.apache.geode.distributed.internal.membership.gms.membership.GMSJoinLeave$ViewCreator.run(GMSJoin
Leave.java:2335)
```

Log Level: error

Category: Operations

Meaning:

The member has been disconnected and forced out of the distributed system. Reasons for disconnection can include: - A network error resulting in a network partition or failure to communicate. - High resource usage (memory or CPU) resulting in the member being ‘suspect’ and eventually being removed

Potential Resolutions:

Examine the ‘System metrics’ logged in the message to determine if the system was under excessive load. Additional warning or error messages in the logs, preceding this event, should also be examined to determine a root cause.

Developing with Tanzu GemFire

This topic provides links to documentation that explains main concepts of application programming with VMware Tanzu GemFire. These topics describe how to plan and implement regions, data serialization, event handling, delta propagation, transactions, and more.

For information about Tanzu GemFire REST application development, see Developing REST Applications for VMware Tanzu GemFire.

- Region Data Storage and Distribution

  The Tanzu 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 Tanzu GemFire before you start configuring your data regions.

- Partitioned Regions

  In addition to basic region management, partitioned regions include options for high availability, data location control, and data balancing across the cluster.

- 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 Tanzu GemFire members.

- Consistency for Region Updates

  Tanzu GemFire ensures that all copies of a region eventually reach a consistent state on all members and clients that host the region, including Tanzu GemFire members that distribute region events.

- 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.

- Data Serialization

  Data that you manage in Tanzu GemFire must be serialized and deserialized for storage and transmittal between processes. You can choose among several options for data serialization.

- Events and Event Handling

  Tanzu GemFire provides versatile and reliable event distribution and handling for your cached data and system member events.

- 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.

- Querying

  Tanzu GemFire provides a SQL-like querying language, Object Query Language (OQL), that allows you to access data stored in Tanzu GemFire regions.

- Continuous Querying

Continuous querying continuously returns events that match the queries you set up.

- Transactions

  Tanzu GemFire provides a transactions API, with begin, commit, and rollback methods. These methods are much the same as the familiar relational database transactions methods.

- Function Execution

  A function is a body of code that resides on a server and that an application can invoke from a client or from another server without the need to send the function code itself. The caller can direct a data-dependent function to operate on a particular dataset, or can direct a data-independent function to operate on a particular server, member, or member group.

Region Data Storage and Distribution

This topic links to documentation about region data storage and distribution in VMware Tanzu GemFire.

The Tanzu GemFire data storage and distribution models put your data in the right place at the right time. The following topics provide information about the options for data storage in Tanzu GemFire.

- Storage and Distribution Options

  Tanzu GemFire provides several models for data storage and distribution, including partitioned or replicated regions as well as distributed or non-distributed regions (local cache storage).

- Region Types

  Region types define region behavior within a single cluster. You have various options for region data storage and distribution.

- Region Data Policies

  Lists and describes the region data policies in Tanzu GemFire.

- Region Data Stores and Data Accessors

  Understand the difference between members that store data for a region and members that act only as data accessors to the region.

Storage and Distribution Options

This topic explains data storage and distribution options in VMware Tanzu GemFire.

Tanzu GemFire provides several models for data storage and distribution, including partitioned or replicated regions as well as distributed or non-distributed regions (local cache storage).

Peer-to-Peer Region Storage and Distribution

At its most general, data management means having current data available when and where your applications need it. In a properly configured Tanzu GemFire installation, you store your data in your local members and Tanzu 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 using the gfsh cluster configuration service, cache.xml file or the API. Once configured, Tanzu 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 cluster and between clients and servers.

- 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.

Data storage in any type of installation is based on the peer-to-peer configuration for each individual cluster. Data and event distribution is based on a combination of the peer-to-peer and system-to-system configurations.

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 refid with a RegionShortcut or ClientRegionShortcut that has local state. These automatically set the region data-policy to a non-empty policy. 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

In VMware Tanzu GemFire, region types define region behavior within a single cluster. You have various options for region data storage and distribution.

Within a Tanzu GemFire cluster, you can define distributed regions and non-distributed regions, and you can define regions whose data is spread across the cluster, 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 cluster. 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.

When you configure a server or peer region using gfsh or with 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.

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 • Very large data sets • High availability • Write performance • Partitioned event listeners and data loaders
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 • Read heavy, small datasets • Asynchronous distribution • Query performance
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, with some members holding replicas and some holding non-replicas.	Peer regions, but not server regions and not client regions • Asynchronous distribution • Query performance
Non-distributed (local)	The region is visible only to the defining member.	Client regions and peer regions • Data that is not shared between applications

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.

> 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. Tanzu 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 redundant copies of your data that Tanzu GemFire should make. If a member fails, your data will be available without interruption from the remaining members that host a redundant copy of the data. No data loss occurs as long as the number of server failures does not exceed the number of redundant copies. 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 is 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 colocate keys within buckets and across multiple partitioned regions. You can also control which members store which data buckets.

Replicated Regions

Replicated regions provide the highest performance in terms of throughput and latency. Replication is a good choice for small to medium size server regions.

Use replicated regions for:

- Small amounts of data required by all members of the cluster. 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 stores only data in which it has expressed an interest, 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 need only notification and updates for changes to a subset of the data. In non-replicated regions, each member receives only 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 Policies

This topic lists and describes the region data policies in VMware Tanzu GemFire.

The region data policy is a region attribute, data-policy. For information about region attributes, see Managing Region Attributes.

A region's data-policy specifies how the local cache handles data for a region. This setting controls behavior such as local data storage and region initialization.

> You can 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.

You can specify a region's data-policy as one of the following:

Data Policy	Description
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 subscription-attributes interest-policy to all.
normal	Data that is used locally (accessed with get, stored with put, etc.) is stored in the local cache. This policy allows the contents in the cache to differ from other caches.
partition	Data is partitioned across local and remote caches using the automatic data distribution behavior of partitioned regions. Additional configuration is done in the partition-attributes.
persistent-partition	Behaves the same as partition and also persists data to disk.
persistent-replicate	Behaves the same as replicate and also persists data to disk.
preloaded	Initializes like a replicated region. After initialization, behaves like a normal region.
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 scope, behaving the same as for normal.

Region Data Stores and Data Accessors

This topic explains the difference between members that store data for a region and members that act only as data accessors to the region in VMware Tanzu GemFire.

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.

For server regions, suppress local data storage at region creation by specifying a region shortcut that contains the term "PROXY" in its name, such as PARTITION_PROXY or REPLICATE_PROXY.

For client regions, suppress local data storage at region creation by specifying the PROXY region shortcut. Do not use the CACHING_PROXY shortcut for this purpose, as it allows local data storage.

Partitioned Regions

This topic provides an overview of VMware Tanzu GemFire partitioned regions and links to documentation that explains partitioned regions in greater detail.

In addition to basic region management, partitioned regions include options for high availability, data location control, and data balancing across the cluster.

- Understanding Partitioning

  To use partitioned regions, you should understand how they work and your options for managing them.

- Configuring Partitioned Regions

  Plan the configuration and ongoing management of your partitioned region for host and accessor members and configure the regions for startup.

- Configuring the Number of Buckets for a Partitioned Region

  Decide how many buckets to assign to your partitioned region and set the configuration accordingly.

- Custom-Partitioning and Colocating Data

  You can customize how VMware Tanzu GemFire groups your partitioned region data with custom partitioning and data colocation.

- Configuring High Availability for Partitioned Regions

  By default, VMware Tanzu GemFire stores only a single copy of your partitioned region data among the region's data stores. You can configure Tanzu GemFire to maintain redundant copies of your partitioned region data for high availability.

- Configuring Single-Hop Client Access to 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.

- Rebalancing Partitioned Region Data

  In a cluster 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.

- Checking Redundancy in Partitioned Regions

Under some circumstances, it can be important to verify that your partitioned region data is redundant and that upon member restart, redundancy has been recovered properly across partitioned region members.

- Moving Partitioned Region Data to Another Member

  You can use the PartitionRegionHelper moveBucketByKey and moveData methods to explicitly move partitioned region data from one member to another.

Understanding Partitioning

This topic explains how region partitioning works in VMware Tanzu GemFire and the options for managing partitioned regions.

During operation, a partitioned region appears to be a single large virtual region, with the same logical view held by all of the members where the region is defined.
Logical View compared to Physical view. Logical view has X, Y, and Z in Partitioned Region A. Physical view has X in Partitioned Region A on Machine 1, Y in Partitioned Region A on Machine 2, and Z in Partitioned Region A on Machine 3.

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.
Logical View compared to Physical view. Logical view has X, Y, and Z in Partitioned Region A. Physical view has no local data in Partitioned Region A on Machine 1, X and Z in Partitioned Region A on Machine 2, and Y in Partitioned Region A on Machine 3.

A cluster 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

Tanzu GemFire automatically determines the physical location of data in the members that host a partitioned region's data. Tanzu 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 Tanzu 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).
- Tanzu 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 cluster.
- In rebalancing data for the region, Tanzu 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 for more information.

Configuring Partitioned Regions

This topic explains how to plan the configuration and ongoing management of your VMware Tanzu GemFire partitioned regions for host and accessor members and configure the regions for startup.

Before you begin, read Basic Configuration and Programming.

1. Start your region configuration using one of the PARTITION region shortcut settings. See Region Shortcuts and Custom Named Region Attributes.
2. If you need high availability for your partitioned region, configure for that. See Configure High Availability for a Partitioned Region.
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.
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.
5. Configure your members' data storage and data loading for the region:
    1. 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. 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.
    2. For members that store data for the region (local-max-memory greater than 0), define a data loader. See Implement a Data Loader.

3. 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 colocate data between multiple regions, code and configure accordingly. See Understanding Custom Partitioning and Data Colocation.
7. Plan your partition rebalancing strategy and configure and program for that. See Rebalancing Partitioned Region Data.

> To configure a partitioned region using gfsh, see gfsh Command Help.

Configuring the Number of Buckets for a Partitioned Region

This topic explains how to configure the number of buckets to assign to your partitioned region in VMware Tanzu GemFire.

The total number of buckets for the partitioned region determines the granularity of data storage and thus how evenly the data can be distributed. Tanzu 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");
```

- gfsh:

  Use the --total-num-buckets parameter of the create region command. For example:

```
gfsh>create region --name="PR1" --type=PARTITION --total-num-buckets=7
```

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.

Region with three members (M1, M2, M3) as described.

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.

Region with more buckets for the three members (M1, M2, M3) as described.

Custom-Partitioning and Colocating Data

This topic explains how you can customize the way VMware Tanzu GemFire groups your partitioned region data with custom partitioning and data colocation.

- Understanding Custom Partitioning and Data Colocation

  Custom partitioning and data colocation can be used separately or in conjunction with one another.

- Standard Custom Partitioning

  By default, Tanzu 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 by providing a standard partition resolver that maps entries to a set of buckets called a partition.

- Fixed Custom Partitioning

  Fixed custom partitioning is deprecated and will be removed in upcoming releases. By default, Tanzu 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 by providing a fixed partition resolver that not only maps entries to a set of buckets called a partition, but also specifies which members host which data buckets.

- Colocate Data from Different Partitioned Regions

  By default, Tanzu 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. Colocation is required for some operations, and it increases performance for others by reducing the number of data accesses to entries that are hosted on other cluster members.

Understanding Custom Partitioning and Data Colocation

This topic explains custom partitioning and data colocation can be used separately or in conjunction with each other in VMware Tanzu GemFire.

Custom Partitioning

Use custom partitioning to group like entries into region buckets within a region. By default, Tanzu GemFire assigns new entries to buckets based on the entry key's hash code. 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 also takes advantage of custom partitioning.

With custom partitioning, you have two choices:

- Standard custom partitioning. With standard custom partitioning, you group entries into buckets, but you do not specify where the buckets reside. Tanzu GemFire always keeps the entries in the buckets you have specified, but may move the buckets around for load balancing. See Standard Custom Partitioning for implementation and configuration details.
- Fixed custom partitioning. Fixed custom partitioning is deprecated and will be removed in upcoming releases. With fixed custom partitioning, you specify the exact member where each region entry resides. You assign an entry to a partition and then to a bucket within that partition. You name 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:

  - Tanzu 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 a 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.

  See Fixed Custom Partitioning for implementation and configuration details.

Data Colocation Between Regions

With data colocation, Tanzu GemFire stores entries that are related across multiple data regions in a single member. Tanzu GemFire does this by storing all of the regions' buckets with the same ID together in the same member. During rebalancing operations, Tanzu 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 colocation 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 colocation where the data is partitioned by customer type.

Member 1 with Partition Regions A and B, and customer X data in each partition region. Member 2 with Partition Regions A and B, customer P data in each partition region

Data colocation requires the same data partitioning mechanism for all of the colocated regions. You can use the default partitioning provided by Tanzu GemFire or any of the custom partitioning strategies.

You must use the same high availability settings across your colocated regions.

See Colocate Data from Different Partitioned Regions for implementation and configuration details.

Custom-Partition Your Region Data

This topic explains how to customize the partitioning of your region data in VMware Tanzu GemFire.

By default, VMware Tanzu 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 by providing your own partition resolver. The partitioning can go further with a fixed-partition resolver that specifies which members host which data buckets.

> If you are both colocating region data and custom partitioning, all colocated regions must use the same custom partitioning mechanism. See Colocate Data from Different Partitioned Regions.

To custom-partition your region data, follow two steps:

1. Implement the interface.
2. Configure the regions.

These steps differ based on which partition resolver is used.

Implementing Standard Partitioning

- Implement the org.apache.geode.cache.PartitionResolver interface within one of the following locations, listed here in the search order used by Tanzu GemFire:
  - Within a custom class. Specify this class as the partition resolver during region creation.
  - Within the key's class. For keys implemented as objects, define the interface within the key's class.
  - Within the cache callback class. Implement the interface within a cache callback's class. When using this implementation, any and all Region operations must be those that specify the callback as a parameter.
- Implement the resolver's getName, init, and close methods.

  A simple implementation of getName is

  ```
  return getClass().getName();
  ```

  The init method does any initialization steps upon cache start that relate to the partition resolver's task.

  The close method accomplishes any clean up that must be accomplished before a cache close completes. For example, close might close files or connections that the partition resolver opened.
- Implement the resolver's getRoutingObject method to return the routing object for each entry. A hash of that returned routing object determines the bucket. Therefore, getRoutingObject should return an object that, when run through its hashCode, directs grouped objects to the desired bucket.

  > Only fields on the key should be used when creating the routing object. Do not use the value or additional metadata for this purpose.

For example, here is an implementation on a region key object that groups using the sum of a 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;
  }
}
```

Implementing the String-Based Partition Resolver

The implementation of a string-based partition resolver is in org.apache.geode.cache.util.StringPrefixPartitionResolver. It does not require any further implementation.

Implementing Fixed Partitioning

- Implement the org.apache.geode.cache.FixedPartitionResolver interface within one of the following locations, listed here in the search order used by Tanzu GemFire:
  - Custom class. Specify this class as the partition resolver during region creation.
  - Entry key. For keys implemented as objects, define the interface for the key's class.
  - Within the cache callback class. Implement the interface within a cache callback's class. When using this implementation, any and all Region operations must be those that specify the callback as a parameter.
- Implement the resolver's getName, init, and close methods.

  A simple implementation of getName is

  ```
  return getClass().getName();
  ```

  The init method does any initialization steps upon cache start that relate to the partition resolver's task.

  The close method accomplishes any clean up that must be accomplished before a cache close completes. For example, close might close files or connections that the partition resolver opened.
- Implement the resolver's getRoutingObject method to return the routing object for each entry. A hash of that returned routing object determines the bucket within a partition.

  This method can be empty for fixed partitioning where there is only one bucket per partition. That implementation assigns partitions to servers such that the application has full control of grouping entries on servers.

  > Only fields on the key should be used when creating the routing object. Do not use the value or additional metadata for this purpose.

- Implement the getPartitionName method to return the name of the partition for each entry, based on where you want the entries to reside. All entries within a partition will be on a single server.

  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() {
  }
}
```

Configuring Standard Partitioning

- Configure the region so Tanzu GemFire finds your resolver for all region operations. How you do this depends on where you chose to implement your custom partitioning.
  - Custom class. Define the class for the region at creation. Use one of these methods:

    XML:

```
<region name="trades">
  <region-attributes>
    <partition-attributes>
      <partition-resolver>
        <class-name>myPackage.TradesPartitionResolver</class-name>
      </partition-resolver>
    </partition-attributes>
  </region-attributes>
</region>
```

Java API:

```
PartitionResolver resolver = new TradesPartitionResolver();
PartitionAttributes attrs =
  new PartitionAttributesFactory()
    .setPartitionResolver(resolver).create();

Cache c = new CacheFactory().create();

Region r = c.createRegionFactory()
  .setPartitionAttributes(attrs)
  .create("trades");
```

gfsh:

Add the option --partition-resolver to the gfsh create region command, specifying the package and class name of the custom partition resolver.

- Entry key. Use the key object with the resolver implementation for every entry operation.
- 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 colocated data is in a server system, add the PartitionResolver implementation class to the CLASSPATH of your Java clients. For Java single-hop access to work, the resolver class needs to have a zero-argument constructor, and the resolver class must not have any state; the init method is included in this restriction.

Configuring the String-Based Partition Resolver

- 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="customers">
  <region-attributes>
    <partition-attributes>
      <partition-resolver>
        <class-name>org.apache.geode.cache.util.StringPrefixPartitionResolver
        </class-name>
      </partition-resolver>
    </partition-attributes>
  </region-attributes>
</region>
```

  Java API:

```
PartitionAttributes attrs =
  new PartitionAttributesFactory()
  .setPartitionResolver("StringPrefixPartitionResolver").create();

Cache c = new CacheFactory().create();

Region r = c.createRegionFactory()
  .setPartitionAttributes(attrs)
  .create("customers");
```

gfsh:

Add the option --partition-resolver=org.apache.geode.cache.util.StringPrefixPartitionResolver to the gfsh create region command.

- For colocated data, add the StringPrefixPartitionResolver implementation class to the CLASSPATH of your Java clients. The resolver will work with Java single-hop clients.

Configuring Fixed Partitioning

Fixed custom partitioning is deprecated and will be removed in upcoming releases.

- 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 org.apache.geode.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.

  > 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>
              <class-name>myPackage.QuarterFixedPartitionResolver</class-name>
            </partition-resolver>
            <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>
    ```

  - 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);

Cache c = new CacheFactory().create();

Region r = c.createRegionFactory()
  .setPartitionAttributes(paf.create())
  .create("Trades");
```

    - gfsh:

      You cannot specify a fixed partition resolver using gfsh.

- If your colocated data is in a server system, add the class that implements the FixedPartitionResolver interface to the CLASSPATH of your Java clients. For Java single-hop access to work, the resolver class needs to have a zero-argument constructor, and the resolver class must not have any state; the init method is included in this restriction.

Standard Custom Partitioning

This topic explains standard custom partitioning in VMware Tanzu GemFire.

By default, Tanzu 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 by providing a standard custom partition resolver that maps entries in a custom manner.

> If you are both colocating region data and custom partitioning, all colocated regions must use the same custom partitioning mechanism. See Colocate Data from Different Partitioned Regions.

To custom-partition your region data, follow two steps:

- implement the interface
- configure the regions

Implementing Standard Custom Partitioning

- Implement the org.apache.geode.cache.PartitionResolver interface in one of the following ways, listed here in the search order used by Tanzu GemFire:
    - Using a custom class. Implement the PartitionResolver within your class, and then specify your class as the partition resolver during region creation.
    - Using the key's class. Have the entry key's class implement the PartitionResolver interface.
    - Using the callback argument's class. Have the class that implements your callback argument implement the PartitionResolver interface. When using this implementation, any and all Region operations must be those that specify the callback argument.

- Implement the resolver's getName, init, and close methods.

  A simple implementation of getName is

  ```
  return getClass().getName();
  ```

  The init method does any initialization steps upon cache start that relate to the partition resolver's task. This method is only invoked when using a custom class that is configured by gfsh or by XML (in a cache.xml file).

  The close method accomplishes any clean up that must be accomplished before a cache close completes. For example, close might close files or connections that the partition resolver opened.

- Implement the resolver's getRoutingObject method to return the routing object for each entry. A hash of that returned routing object determines the bucket. Therefore, getRoutingObject should return an object that, when run through its hashCode, directs grouped objects to the desired bucket.

  > Only the key, as returned by getKey, should be used by getRoutingObject. Do not use the value associated with the key or any additional metadata in the implementation of getRoutingObject. Do not use getOperation or getValue.

Implementing the String Prefix Partition Resolver

Tanzu GemFire provides an implementation of a string-based partition resolver in org.apache.geode.cache.util.StringPrefixPartitionResolver. This resolver does not require any further implementation. It groups entries into buckets based on a portion of the key. All keys must be strings, and each key must include a '|' character, which delimits the string. The substring that precedes the '|' delimiter within the key will be returned by getRoutingObject.

Configuring the Partition Resolver Region Attribute

Configure the region so Tanzu GemFire finds your resolver for all region operations.

- Custom class. Use one of these methods to specify the partition resolver region attribute:

  gfsh:

  Add the option --partition-resolver to the gfsh create region command, specifying the package and class name of the standard custom partition resolver.

  gfsh for the String Prefix Partition Resolver:

  Add the option --partition-resolver=org.apache.geode.cache.util.StringPrefixPartitionResolver to the gfsh create region command.

  Java API:

  ```
  PartitionResolver resolver = new TradesPartitionResolver();
  PartitionAttributes attrs =
     new PartitionAttributesFactory()
     .setPartitionResolver(resolver).create();

  Cache c = new CacheFactory().create();

  Region r = c.createRegionFactory()
     .setPartitionAttributes(attrs)
     .create("trades");
  ```

  Java API for the String Prefix Partition Resolver:

```
PartitionAttributes attrs =
   new PartitionAttributesFactory()
   .setPartitionResolver(new StringPrefixPartitionResolver()).create();

Cache c = new CacheFactory().create();

Region r = c.createRegionFactory()
   .setPartitionAttributes(attrs)
   .create("customers");
```

XML:

```
<region name="trades">
   <region-attributes>
      <partition-attributes>
         <partition-resolver>
            <class-name>myPackage.TradesPartitionResolver
            </class-name>
         </partition-resolver>
      <partition-attributes>
   </region-attributes>
</region>
```

XML for the String Prefix Partition Resolver:

```
<region name="customers">
   <region-attributes>
      <partition-attributes>
         <partition-resolver>
            <class-name>org.apache.geode.cache.util.StringPrefixPartitionResolver
            </class-name>
         </partition-resolver>
      <partition-attributes>
   </region-attributes>
</region>
```

- If your colocated data is in a server system, add the PartitionResolver implementation class to the class path of your Java clients. For Java single-hop access to work, the resolver class needs to have a zero-argument constructor, and the resolver class must not have any state; the init method is included in this restriction.

Fixed Custom Partitioning

Fixed custom partitioning is deprecated and will be removed in upcoming releases.

This topic explains fixed custom partitioning in VMware Tanzu GemFire.

By default, Tanzu 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 by providing a fixed custom partition resolver that not only maps entries to a set of buckets called a partition, but also specifies which members host which data buckets.

> If you are both colocating region data and custom partitioning, all colocated regions must use the same custom partitioning mechanism. See Colocate Data from Different Partitioned Regions.

To custom-partition your region data, follow two steps:

- implement the interface
- configure the regions

These steps differ based on which partition resolver is used.

Implementing Fixed Custom Partitioning

- Implement the org.apache.geode.cache.FixedPartitionResolver interface within one of the following locations, listed here in the search order used by Tanzu GemFire:
    - Custom class. Specify this class as the partition resolver during region creation.
    - Entry key. For keys implemented as objects, define the interface for the key's class.
    - Within the cache callback class. Implement the interface within a cache callback's class. When using this implementation, any and all Region operations must be those that specify the callback as a parameter.
- Implement the resolver's getName, init, and close methods.

  A simple implementation of getName is

  ```
  return getClass().getName();
  ```

  The init method does any initialization steps upon cache start that relate to the partition resolver's task.

  The close method accomplishes any clean up that must be accomplished before a cache close completes. For example, close might close files or connections that the partition resolver opened.
- Implement the resolver's getRoutingObject method to return the routing object for each entry. A hash of that returned routing object determines the bucket within a partition.

  This method can be empty for fixed partitioning where there is only one bucket per partition. That implementation assigns partitions to servers such that the application has full control of grouping entries on servers.

  > Only fields on the key should be used when creating the routing object. Do not use the value or additional metadata for this purpose.

- Implement the getPartitionName method to return the name of the partition for each entry, based on where you want the entries to reside. All entries within a partition will be on a single server.

  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() {
  }
}
```

Configuring Fixed Custom Partitioning

- 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 org.apache.geode.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.

> 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>
          <class-name>myPackage.QuarterFixedPartitionResolver</class-name>
        </partition-resolver>
        <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>
```

- 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);

Cache c = new CacheFactory().create();

Region r = c.createRegionFactory()
   .setPartitionAttributes(paf.create())
   .create("Trades");
```

- gfsh:

  You cannot specify a fixed partition resolver using gfsh.

- If your colocated data is in a server system, add the class that implements the FixedPartitionResolver interface to the CLASSPATH of your Java clients. For Java single-hop access to work, the resolver class needs to have a zero-argument constructor, and the resolver class must not have any state; the init method is included in this restriction.

Colocate Data from Different Partitioned Regions

This topic explains how to colocate data from different partitioned regions in VMware Tanzu GemFire.

By default, VMware Tanzu 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.

Colocation is required for some operations, and it increases performance for others by reducing the number of data accesses to entries that are hosted on other cluster members.

Data colocation 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.

Procedure

1. Identify one region as the central region, with which data in the other regions is explicitly colocated. If you use persistence for any of the regions, you must persist the central region.
    1. 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 colocated regions in the XML, you must create the central region in the XML before the others. Tanzu GemFire will verify its existence when the others are created and return IllegalStateException if the central region is not there. Do not add any colocation specifications to this central region.
    2. 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>
    </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");
```

        - gfsh:

```
gfsh>create region --name="trades" type=PARTITION
gfsh> create region --name="trade_history" --colocated-with="trades"
```

2. For each of the colocated 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, specify the custom resolver for all colocated regions. This example uses the same partition resolver for both regions:
   - XML:

     ```
     <cache>
       <region name="trades">
         <region-attributes>
           <partition-attributes>
             <partition-resolver name="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 name="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");
     ```

   - gfsh:

     Specify a partition resolver as described in the configuration section of Custom-Partition Your Region Data.
4. If you want to persist data in the colocated regions, persist the central region and then persist the other regions as needed. Use the same disk store for all of the colocated regions that you persist.

Configuring High Availability for Partitioned Regions

This topic links to documentation that explains how to configure high-availability for partitioned regions in VMware Tanzu GemFire.

By default, Tanzu GemFire stores only a single copy of your partitioned region data among the region's data stores. You can configure Tanzu 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.

- 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.

Understanding High Availability for Partitioned Regions

This topic explains high availability for partitioned regions in VMware Tanzu GemFire.

With high availability, each member that hosts data for the partitioned region receives 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, Tanzu 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.

> 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. Use the gemfire.MAX_PARALLEL_BUCKET_RECOVERIES system property to configure the maximum number of buckets that are recovered in parallel. By default, up to 8 buckets are recovered in parallel any time the system attempts to recover redundancy.

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, Tanzu 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, Tanzu GemFire places copies on the same physical machine. You can change this behavior, so Tanzu GemFire only places copies on separate machines.

You can also control which members store your primary and secondary data copies. Tanzu GemFire provides two options:

- Fixed custom partitioning. Fixed custom partitioning is deprecated and will be removed in upcoming releases. 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 Tanzu 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 Tanzu 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, Tanzu GemFire will not place two copies of the data in the same zone, so make sure you have enough zones.

Running Processes in Virtual Machines

By default, Tanzu GemFire stores redundant copies on different machines. When you run your processes in virtual machines, the normal view of the machine becomes the VM and not the physical machine. If you run multiple VMs on the same physical machine, you could end up storing partitioned region primary buckets in separate VMs, but on the same physical machine as your secondaries. If the physical machine fails, you can lose data. When you run in VMs, you can configure Tanzu GemFire to identify the physical machine and store redundant copies on different physical machines.

Reads and Writes in Highly-Available Partitioned Regions

Tanzu 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 does not have a local copy of Y or Z, it goes to a cache that does, picking the source cache at random.

Reads and writes in highly-available partitioned regions as described above.

Configure High Availability for a Partitioned Region

This topic explains how to configure high availability for a partitioned regions in VMware Tanzu GemFire.

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.
2. (Optional) If you want to group your data store members into redundancy zones, configure them accordingly. See Configure Redundancy Zones for Members.
3. (Optional) If you want Tanzu GemFire to only place redundant copies on different physical machines, configure for that. See Set Enforce Unique Host.
4. Decide how to manage redundancy recovery and change Tanzu GemFire's default behavior as needed.
    - 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.
    - 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.
5. Decide how many buckets Tanzu GemFire should attempt to recover in parallel when performing redundancy recovery. By default, the system recovers up to 8 buckets in parallel. Use the gemfire.MAX_PARALLEL_BUCKET_RECOVERIES system property to increase or decrease the maximum number of buckets to recover in parallel any time redundancy recovery is performed.
6. 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.

For all partitioned regions, redundancy can be restored without moving buckets between members using a restore redundancy operation. See Restoring Redundancy in Partitioned Regions.

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.

- Configure Redundancy Zones for Members

  Group members into redundancy zones so Tanzu GemFire will separate redundant data copies into different zones.

- Set Enforce Unique Host

  Configure Tanzu GemFire to use only unique physical machines for redundant copies of partitioned region data.

- Configure Member Crash Redundancy Recovery for a Partitioned Region

  Configure whether and how redundancy is recovered in a partition region after a member crashes.

- Configure Member Join Redundancy Recovery for a Partitioned Region

  Configure whether and how redundancy is recovered in a partition region after a member joins.

Set the Number of Redundant Copies

This topic explains how to set the number of redundant copies of your partitioned region data in VMware Tanzu GemFire.

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();
```

- gfsh:

```
gfsh>create region --name="PR1" --type=PARTITION --redundant-copies=1
```

Configure Redundancy Zones for Members

This topic explains how to configure redundancy zones for members in VMware Tanzu GemFire.

By grouping members into redundancy zones. Tanzu GemFire separates redundant data copies into different zones.

For more information about setting member's gemfire.properties settings, see Reference.

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

This topic explains how to configure VMware Tanzu GemFire to use only unique physical machines for redundant copies of partitioned region data.

For more information about setting a member's gemfire.properties settings, see Reference.

Configure your members so Tanzu 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

This topic explains how to configure whether and how redundancy is recovered in a VMware Tanzu GemFire 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();
```

- gfsh:

```
gfsh>create region --name="PR1" type=PARTITION --recovery-delay=10000
```

Configure Member Join Redundancy Recovery for a Partitioned Region

This topic explains how to configure member join redundancy recovery for partitioned regions in VMware Tanzu GemFire.

Use the partition attribute startup-recovery-delay to specify member join redundancy recovery.

value of startup-recovery-delay	Effect following a member join
-1	No automatic recovery of redundancy after a new member comes online. With this value and the default recovery-delay setting, redundancy recovery is only achieved by a rebalance operation.
long >= 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 member that hosts the partitioned region joins.

Setting startup-recovery-delay to a value higher than the default of 0 allows multiple new members to join before redundancy recovery begins. 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.

> Satisfying redundancy is not the same as adding capacity. If redundancy is satisfied, new members do not take buckets until the invocation of a rebalance operation.

The parallel recovery implementation recovers quickly. For this reason, it is even more important to configure startup-recovery-delay to an appropriate value when restarting multiple members at the same time. Set startup-recovery-delay to a value that ensures all members are up and available *before* redundancy recovery kicks in.

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();
```

- gfsh:

```
gfsh>create region --name="PR1" --type=PARTITION --startup-recovery-delay=5000
```

Configuring Single-Hop Client Access to Partitioned Regions

This topic explains how to configure client single-hop access to partitioned regions on a VMware Tanzu GemFire server.

Client connection pools support direct, single-hop access to partitioned region data in the servers by means of a parameter, pr-single-hop-enabled, which is true by default for any newly-created connection pool.

Understanding Client Single-Hop Access to Partitioned Regions

When enabled, single-hop access is used for the following operations when they are applied to data in partitioned regions:

- put
- get
- destroy
- invalidate
- putAll
- getAll
- removeAll
- onRegion function execution

To perform single-hop data access, clients use metadata from the servers about where the entry buckets are hosted. The metadata is cached locally and maintained 'lazily', that is, it is updated only after a single-hop operation attempt requires multiple hops to succeed, an indicator of stale metadata in the client.

How Single-Hop and Connection Limit Settings Interact

The connection pool supports two sets of connection limits, a maximum and minimum limit per server, used when single-hop mode is in effect, and a maximum and minimum limit for the pool as a whole, used when single-hop mode is disabled:

Parameter	Default Value	Notes
max-connections-per-server	-1 (no maximum)	Effective when pr-single-hop-enabled=true
min-connections-per-server	1	Effective when pr-single-hop-enabled=true
max-connections	-1 (no maximum)	Effective when pr-single-hop-enabled=false
min-connections	1	Effective when pr-single-hop-enabled=false

When pr-single-hop-enabled is true,

- max-connections-per-server and min-connections-per-server are used to configure the client pool.
- In this case, max-connections and min-connections are customarily unused (that is, set to their default values), and are ignored.
- If max-connections and min-connections are set to non-default values (for example, if you are updating a pre-v10.1 client application), their values are converted to max-connections-per-server and min-connections-per-server values using the Connection Limit Adaptation Formula.
    - This conversion is provided as a courtesy; it is not intended as a long-term solution.
    - For production systems, set max-connections-per-server and min-connections-per-server values explicitly and reset max-connections and min-connections to their default values.

When pr-single-hop-enabled is false,

- max-connections and min-connections are used to configure the client pool.
- Default settings of max-connections-per-server and min-connections-per-server are ignored.
- If your application attempts to set max-connections-per-server or min-connections-per-server in this case, an exception is thrown.

How Single-Hop and Connection Timeout Settings Interact

The connection pool supports two timeout parameters:

Parameter	Default Value
server-connection-timeout	0
free-connection-timeout	10000 ms (10 seconds)

When pr-single-hop-enabled is true

- If the max-connections-per-server parameter is set to its default value of -1, meaning “no maximum”, then timeouts are ignored. The connection request results immediately in a connection or in a “NoAvailableServerException”.
- If the max-connections-per-server parameter is set to a value other than its default, then:
    - The single-hop operations listed above first try to get a direct connection to the server hosting the data. The single-hop request is subject to both timeout intervals:
    - The operation tries to get a direct connection to the desired server until server-connection-timeout (default 0) has elapsed.
    - If no direct connection is obtained in time, the client then tries to get a connection to any server until free-connection-timeout (default 10 seconds) has elapsed.
    - If no connection is established when free-connection-timout expires, an “AllConnectionsInUseException” is thrown.
    - You can tune these two timeout periods to suit your application’s needs, which may be especially useful if your application has a low max-connections-per-server count.
    - The two timeout periods are additive. This means that the maximum time that a single-hop operation will spend trying to get a connection is the sum of the two.

When pr-single-hop-enabled is false

- If the max-connections parameter is set to its default value of -1, meaning “no maximum”, then timeouts are ignored. The connection request results immediately in a connection or in a “NoAvailableServerException”.
- If the max-connections parameter is set to a value other than its default, then:
    - The client tries to get a connection to any server until free-connection-timeout (default 10 seconds) has elapsed.
    - If no connection is established when free-connection-timout expires, an “AllConnectionsInUseException” is thrown.
    - The server-connection-timeout parameter does not apply – when single-hop is not in effect, the server-specific timeout is ignored.

Optimizing Single-Hop Access

To make best use of single-hop capabilities:

1. Verify that the client’s pool attribute, pr-single-hop-enabled is not set or is set to true. pr-single-hop-enabled is set to true by default.
2. 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. 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.

3. 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.

Balancing Single-Hop Server Connection Use

Single-hop provides the greatest 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 deactivate single hop and go through servers that do not host the data.

Performance Tuning Considerations

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 deactivate single hop by setting the pool attribute pr-single-hop-enabled to false in your pool declarations.

Without single-hop, the client uses the first server connection available. The server that receives the request determines the data location and contacts the host, which might be a different server, so additional requests are made to the server system.

Setting no limit on connections 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 can accommodate it.

Connection Limit Adaptation Formula

The max-connections-per-server and min-connections-per-server parameters were introduced in Tanzu GemFire version 10.1. They take effect when pr-single-hop-enabled is true, which is the default. This means that client applications developed before these per-server connection limits existed (Tanzu GemFire versions 10.0 and earlier) must be updated to use the new per-server limits.

If your client connection pool runs in single-hop mode (pr-single-hop-enabled is true, which is the default), then you should:

- Set max-connections and min-connections to their default values of -1 and 1, respectively.
- Use max-connections-per-server and min-connections-per-server parameters to configure your connection pool.

If GemFire v10.1 (or later) detects that:

- pr-single-hop-enabled=true, and
- max-connections-per-server and min-connections-per-server are set to their default values of -1 and 1, respectively, and
- max-connections and min-connections are NOT set to their default values, then

- GemFire assumes that you are running a legacy application that has not yet been updated to use the new per-server parameters and applies an adaptation formula to convert max-connections and min-connections, respectively, to max-connections-per-server and min-connections-per-server.

This conversion is provided as a courtesy to allow older applications to continue functioning until you reset the connection limits. It is not intended as a long-term solution.

When GemFire converts max-connections to max-connections-per-server (or convert min-connections to min-connections-per-server), it uses the ceiling of max-connections (or min-connections) divided by the current number of servers known to the client. For example,

- If current number of servers is 3, and
- max-connections is 10, then
- the resulted max-connections-per-server is 4, so
- each server will have max-connections-per-server=4, and
- the total maximum number of connections for the client pool is 12.

If a fourth server joins later:

- The max-connections-per-server of the fourth server is set as ceilingOf(10/4)=3, so
- the total max number of connections for the client pool becomes 15.

> The conditions described above are most likely to result from running a pre-10.1 client application, but can also result from mis-configuring the client pool connection limits. In either case, you should update your client application to use the new per-server connection limits as soon as possible when pr-single-hop-enabled is true.

Rebalancing Partitioned Region Data

This topic explains how to rebalance partitioned region data in VMware Tanzu GemFire.

In a cluster 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.
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 cluster.

For efficiency, when starting multiple members, trigger the rebalance a single time, after you have added all members.

> 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 cluster. 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 simulate a rebalance operation. Type help rebalance or see rebalance 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("Transfered " + results.getTotalBucketTransferBytes()+ "bytes\n");
  ```

You can also just simulate a rebalance through the API, to see if it is 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.

By default, rebalancing is performed on one partitioned region at a time. For regions that have colocated data, the rebalancing works on the regions as a group, maintaining the data colocation between the regions.

You can optionally rebalance multiple regions in parallel by setting the gemfire.resource.manager.threads system property. Setting this property to a value greater than 1 enables Tanzu GemFire to rebalance multiple regions in parallel, any time a rebalance operation is initiated using the API.

You can optionally tell the system to move multiple buckets in parallel during a rebalance operation by setting the system property gemfire.MAX_PARALLEL_REBALANCE_OPERATIONS to a value greater than its default of 1.

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.

Tanzu 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, shut down or failure.

You may also need to rebalance when 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 colocate your partitioned region data (see Colocate Data from Different Partitioned Regions). 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.

Rebalancing solely for the purpose of restoring lost redundancy, when redundancy is being used for high availability and the region has been configured to not automatically recover redundancy after a loss, is not necessary. Instead, the restore redundancy operation should be triggered. See Restoring Redundancy in Partitioned Regions.

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
```

> 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.

Automated Rebalancing

The experimental automated rebalance feature triggers a rebalance operation based on a time schedule.

Checking Redundancy in Partitioned Regions

This topic explains how to check redundancy in VMware Tanzu GemFire partitioned regions.

Under some circumstances, it can be important to verify that your partitioned region data is redundant and that upon member restart, redundancy has been recovered properly across partitioned region members.

Initiate an operation to report the current redundancy status of regions using one of the following:

- gfsh command. Start gfsh and connect to the cluster. Then type the following command:

  ```
  gfsh>status redundancy
  ```

  Optionally, you can specify regions to include or exclude from restoring redundancy. Type help restore redundancy or see status redundancy for more information.

- API call:

```
ResourceManager manager = cache.getResourceManager();
RestoreRedundancyResults currentStatus = manager.createRestoreRedundancyOperation().redundancyStatus();
//These are some of the details we can get about the run from the API
System.out.println("Status for all regions: " + currentStatus.getMessage());
System.out.println("Number of regions with no redundant copies: " + currentStatus.getZeroRedundancyRegionResults
().size();
System.out.println("Status for region " + regionName + ": " + currentStatus.getRegionResult(regionName).getMessage
();
```

If you have startup-recovery-delay=-1 configured for your partitioned region, you will need to trigger a restore redundancy operation on your region after you restart any members in your cluster in order to recover redundancy. See Restoring Redundancy in Partitioned Regions.

If you have startup-recovery-delay set to a low number, you may need to wait extra time until the region has recovered redundancy.

Restoring Redundancy in Partitioned Regions

This topic explains how to restore redundancy in partitioned regions in VMware Tanzu GemFire.

Restoring redundancy is a member operation. It affects all partitioned regions defined by the member, regardless of whether the member hosts data for the regions.

Restoring redundancy creates new redundant copies of buckets on members hosting the region and by default reassigns which members host the primary buckets to give better load balancing. It does not move buckets from one member to another. The reassignment of primary hosts can be prevented using the appropriate flags, as described below. See Configure High Availability for a Partitioned Region for further detail on redundancy.

For efficiency, when starting multiple members, trigger the restore redundancy a single time, after you have added all members.

Initiate a restore redundancy operation using one of the following:

- gfsh command. First, starting a gfsh prompt and connect to the cluster. Then type the following command:

  ```
  gfsh>restore redundancy
  ```

  Optionally, you can specify regions to include or exclude from restoring redundancy, and prevent the operation from reassigning which members host primary copies. Type help restore redundancy or see restore redundancy for more information.

- API call:

```
ResourceManager manager = cache.getResourceManager();
CompletableFuture<RestoreRedundancyResults> future = manager.createRestoreRedundancyOperation()
  .includeRegions(regionsToInclude)
  .excludeRegions(regionsToExclude)
  .shouldReassignPrimaries(false)
  .start();
//Get the results
RestoreRedundancyResults results = future.get();
//These are some of the details we can get about the run from the API
System.out.println("Restore redundancy operation status is " + results.getStatus());
System.out.println("Results for each included region: " + results.getMessage());
System.out.println("Number of regions with no redundant copies: " + results.getZeroRedundancyRegionResults().size
();
System.out.println("Results for region " + regionName + ": " + results.getRegionResult(regionName).getMessage();
```

If you have startup-recovery-delay=-1 configured for your partitioned region, you will need to perform a restore redundancy operation on your region after you restart any members in your cluster in order to recover any lost redundancy.

If you have startup-recovery-delay set to a low number, you may need to wait extra time until the region has recovered redundancy.

Moving Partitioned Region Data to Another Member

This topic explains how to move partitioned data from one member to another in VMware Tanzu GemFire.

You can use the PartitionRegionHelper moveBucketByKey and moveData methods to explicitly move partitioned region data from one member to another.

The moveBucketByKey method moves the bucket that contains the specified key from a source member to a destination member. For example, you could use the method to move a popular product item to a new, empty member to reduce load on the source member.

For example:

```
Object product = ...
Region r = ...
DistributedSystem ds = ...
String memberName = ...

//Find the member that is currently hosting the product.
Set<DistributedMember> sourceMembers =
PartitionRegionHelper.getAllMembersForKey(r, product);

//Find the member to move the product to.
DistributedMember destination = ds.findDistributedMember(memberName);

//In this example we assume there is always at least one source.
//In practice, you should check that at least one source
//for the data is available.
source = sourceMembers.iterator().next();

//Move the bucket to the new node. The bucket will
//be moved when this method completes. It throws an exception
//if there is a problem or invalid arguments.
PartitionRegionHelper.moveBucketByKey(r, source, destination, product);
```

See the Java API documentation for org.apache.geode.cache.partition.PartitionRegionHelper.moveBucketByKey for more details.

The moveData method moves data up to a given percentage (measured in bytes) from a source member to a destination member. For example, you could use this method to move a specified percentage of data from an overloaded member to another member to improve distribution.

For example:

```
Region r = ...
DistributedSystem ds = ...
String sourceName = ...
String destName = ...

//Find the source member.
DistributedMember source = ds.findDistributedMember(sourceName);
DistributedMember destination = ds.findDistributedMember(destName);

//Move up to 20% of the data from the source to the destination node.
PartitionRegionHelper.moveData(r, source, destination, 20);
```

See the Java API documentation for org.apache.geode.cache.partition.PartitionRegionHelper.moveData for more details.

For more information about partitioned regions and rebalancing, see Partitioned Regions.

Distributed and Replicated Regions

This topic links to documentation about distributed and replicated regions in VMware Tanzu GemFire.

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 Tanzu GemFire members.

- How Distribution Works

  To use distributed and replicated regions, you should understand how they work and your options for managing them.

- 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 Tanzu GemFire members that host the region.

- 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.

- Configure Distributed, Replicated, and Preloaded Regions

  Plan the configuration and ongoing management of your distributed, replicated, and preloaded regions, and configure the regions.

- 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.

How Distribution Works

This topic explains how region distribution and replication work in VMware Tanzu GemFire.

> The management of replicated and distributed regions supplements the general information for managing data regions provided in Basic Configuration and Programming. See also org.apache.geode.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 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 cluster.

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.

Application updating or creating entry X1 in Distributed Region A on Member 1 (M1). No entry in Distributed Region A on Member 2 (M2)

Step 2: The new value is automatically distributed to caches holding the entry.

X1 automatically distributed to Distributed Region A on Member 2 (M2)

Step 3: The entry's value is the same throughout the cluster.

X1 present in Distributed Region A on Member 1 (M2) and Member 2 (M2)

Options for Region Distribution

This topic describes options for region distribution in VMware Tanzu GemFire.

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 Tanzu GemFire members that host the region.

Each distributed region must have the same scope and concurrency checking setting throughout the cluster.

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-no-ack scope has been deprecated and will be removed in a future

release. There is no replacement.

- distributed-ack. Distribution waits for acknowledgment from other caches before continuing. 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 cluster 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 cluster 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 cluster, 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

This topic explains how region replication and preloading work in VMware Tanzu GemFire.

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 cluster.
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 cluster.
4. The union of all copies of the region in the distributed cache.

Member M1 with Distributed Region A containing X and Y. Member M2 with Distributed Region A containing X and Z. Newly created Member M3 with Distributed Region A contains X, Y, and Z.

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.

Distributed Region (Preloaded or Normal) containing X and Y' and 'Distributed System' with double-headed arrow labeled 'Operations for existing entry keys'.

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.

Distributed Region (Replica) containing X and Y' and 'Distributed System'. Arrow labeled 'All creations' from Distributed System to Distributed Region, and double-headed arrow labeled 'Other entry operations (due to distribution) connecting Distributed System to Distributed Region.

Configure Distributed, Replicated, and Preloaded Regions

This topic explains how to plan the configuration and ongoing management of your distributed, replicated, and preloaded regions in VMware Tanzu GemFire, and how to configure regions.

Before you begin, understand Basic Configuration and Programming.

1. Choose the region shortcut setting that most closely matches your region configuration. See org.apache.geode.cache.RegionShortcut or Region Shortcuts. 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>
   ```

   You can also use gfsh to configure a region. For example:

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

   See Region Types.

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

4. If you are using global scope, program any explicit locking you need in addition to the automated locking provided by Tanzu 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

This topic explains locking in global regions in VMware Tanzu GemFire.

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. Tanzu 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 org.apache.geode.cache.Region.getDistributedLock and org.apache.geode.cache.Region.getRegionDistributedLock return instances of java.util.concurrent.locks.Lock for a region and a specified key.

> You must use the Region API to lock regions and region entries. Do not use the DistributedLockService in the org.apache.geode.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>
  ```

  gfsh:

  ```
  gfsh>alter runtime --lock-timeout=60
  ```

- 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>
```

gfsh:

```
gfsh>alter runtime --lock-lease=120
```

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.

   > 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();
  }
```

Consistency for Region Updates

This topic links to documentation about consistency for region updates in VMware Tanzu GemFire.

Tanzu GemFire ensures that all copies of a region eventually reach a consistent state on all members and clients that host the region, including Tanzu GemFire members that distribute region events.

- Consistency Checking by Region Type

  Tanzu GemFire performs different consistency checks depending on the type of region you have configured.

- Configuring Consistency Checking

  Tanzu GemFire enables consistency checking by default. You cannot deactivate consistency checking for persistent regions. For all other regions, you can explicitly activat or deactivate consistency checking by setting the `concurrency-checks-enabled` region attribute in `cache.xml` to `true` or `false`.

- 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.

- How Consistency Checking Works for Replicated Regions

  Each region stores version and timestamp information for use in conflict detection. Tanzu GemFire members use the recorded information to detect and resolve conflicts consistently before applying a distributed update.

- How Destroy and Clear Operations Are Resolved

  When consistency checking is enabled for a region, a Tanzu 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*. Tanzu GemFire retains tombstones for partitioned regions and non-replicated regions as well as for replicated regions, in order to provide consistency.

- 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.

Consistency Checking by Region Type

This topic explains consistency checking by region type in VMware Tanzu GemFire.

Tanzu GemFire performs different consistency checks depending on the type of region you have configured.

Partitioned Region Consistency

For a partitioned region, Tanzu GemFire maintains consistency by routing all updates on a given key to the Tanzu 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 Tanzu GemFire member, all members apply the updates in the same order and consistency is maintained at all times. See Understanding Partitioning.

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, Tanzu 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 and How Destroy and Clear Operations Are Resolved provide more details about how Tanzu GemFire performs conflict checking when applying an update.

Non-Replicated Regions 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.

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 Tanzu GemFire server, which generates the version information necessary for subsequent conflict checking.

Configuring Consistency Checking

Tanzu GemFire enables consistency checking by default. You cannot deactivate consistency checking for persistent regions. For all other regions, you can explicitly activate or deactivate consistency checking by setting the `concurrency-checks-enabled` region attribute in `cache.xml` to “true” or “false.”

All Tanzu GemFire members that host a region must use the same `concurrency-checks-enabled` setting for that region.

A client cache can deactivate consistency checking for a region even if server caches activate 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 unsynchronized with the server cache.

See <region-attributes>.

> 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 entry 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.

If you cannot support the additional overhead in your deployment, you can deactivate consistency checks by setting concurrency-checks-enabled to “false” for each region. See Consistency for Region Updates.

How Consistency Checking Works for Replicated Regions

Each region stores version and timestamp information for use in conflict detection. Tanzu 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 Tanzu 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 Tanzu GemFire members when the local entry is updated.

A Tanzu 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 Tanzu GemFire members updated the same entry at the same time. To resolve a concurrent update, a Tanzu 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.

> When a Tanzu 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. You can track the number of discarded updates for each member using the conflatedEvents statistic. See Tanzu GemFire Statistics List. 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 Tanzu GemFire member. The example below describes this behavior in more detail.

The following example shows how a concurrent update is handled in a cluster of three Tanzu 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 Tanzu 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.

Application update entry X in Distributed Region on Member A while another application update entry X in Distributed Region on Member C

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 the update (A3) is lower than the ID stored in the cache (C3), member C discards the update (and increments the conflatedEvents statistic).

Member A distributes update message to Members B and C. As described above, Member B updates its X, and Member C does not update X.

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.

Member C distributes update message to Members A and B. As described above, Member A updates its X, and Member B updates its X.

How Destroy and Clear Operations Are Resolved

When consistency checking is enabled for a region, a Tanzu 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*. Tanzu 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 Tanzu GemFire member. Garbage collection is automatically triggered after 100,000 tombstones of any type have timed out in the local Tanzu GemFire member. You can optionally set the gemfire.tombstone-gc-threshold property to a value smaller than 100000 to perform garbage collection more frequently.

> To avoid out-of-memory errors, a Tanzu 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 Tanzu GemFire Statistics List.

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, Tanzu 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 normal, preloaded or empty region, the transaction is first delegated to a Tanzu 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 on normal, preloaded or empty regions is that, when consistency checking is enabled, a transaction cannot perform a localDestroy or localInvalidate operation against the region. Tanzu GemFire throws an UnsupportedOperationInTransactionException 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

This topic explains how consistency across regions in WAN deployments is achieved in VMware Tanzu GemFire.

When regions in two or more Tanzu 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.

With a distributed WAN configuration, multiple gateway senders asynchronously queue and send region updates to another Tanzu 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, Tanzu GemFire first ensures that each cluster performs consistency checking to regions before queuing updates to a gateway sender for WAN distribution. In order words, region conflicts are first detected and resolved in the local cluster, using the techniques described in the previous sections.

When a Tanzu 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 Tanzu GemFire clusters. The default conflict checking behavior for WAN-replicated regions is summarized as follows:

- If an update is received from the same Tanzu 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 Tanzu 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.

When a gateway sender distributes an event to another Tanzu 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).

> If you use the default conflict checking feature for WAN deployments, you must ensure that all Tanzu GemFire members in all clusters synchronize their system clocks. For example, use a common NTP server for all Tanzu GemFire members that participate in a WAN deployment.

You can override the default consistency checking for WAN events by installing a custom conflict resolver. Using a custom resolver, installed as a plug-in for a region, 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. 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 “Implementing a GatewayConflictResolver” under Resolving Conflicting Events to configure a custom resolver.

General Region Data Management

This topic links to documentation about general region data management in VMware Tanzu GemFire.

For all regions, you have options to control memory use, back up your data to disk, and discard stale data from 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.

- Eviction

  Use eviction to control data region size. Eviction actions are triggered by space-based thresholds.

- Expiration

  Use expiration to keep data current and to reduce region size by removing stale entries. Expiration actions are triggered by time-based thresholds.

- Keeping the Cache Synchronized with Outside Data Sources

  Keep your distributed cache synchronized with an outside data source by programming and installing application plug-ins for your region.

Persistence and Overflow

This topic links to documentation about data persistence and overflow in VMware Tanzu GemFire.

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.

> This supplements the general steps for managing data regions provided in Basic Configuration and Programming.

All disk storage uses VMware Tanzu GemFire Disk Storage.

- How Persistence and Overflow Work

  To use Tanzu GemFire persistence and overflow, you should understand how they work with your data.

- Configure Region Persistence and Overflow

  Plan persistence and overflow for your data regions and configure them accordingly.

- Overflow Configuration Examples

  The cache.xml examples show configuration of region and server subscription queue overflows.

How Persistence and Overflow Work

This topic explains how data persistence and overflow work in VMware Tanzu GemFire.

Tanzu GemFire persists and overflows several types of data. You can persist or overflow the application data in your regions. In addition, Tanzu GemFire persists and overflows messaging queues, 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 Tanzu GemFire disk stores. For any disk option, you can specify the name of the disk store to use or use the Tanzu GemFire default disk store. See Disk Storage.

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.

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.

Member with Region A containing X, Y, and Z. Disk Files (keys and values for all entries). Arrows labeled 'persist values from create' and 'update/invalidate/destroy' from Member to Disk Files.

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.

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.

Member with Region B containing Y and Z. Disk Files (keys and values for all entries) with X. Arrows labeled 'offload overflow' and 'update/invalidate/destroy' from Member to Disk Files. Arrow from Disk FIles to Member labeled 'satisfy get'.

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.

Member with Region C containing Y and Z. Disk Files (keys and values for all entries) with X, Y, and Z. Arrows labeled 'persist values from create' and 'update/invalidate/destroy' from Member to Disk Files. Arrow from Disk FIles to Member labeled 'satisfy get'.

Persistence and Multi-Site Configurations

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.

Configure Region Persistence and Overflow

This topic explains how to configure data persistence and overflow for your data regions in VMware Tanzu GemFire.

Use the following steps to configure your data regions for persistence and overflow:

1. Configure your disk stores as needed. See Defining and Configuring Disk Stores. 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>
     ```

     Notes: - For partitioned regions, if the partition region attribute local-max-memory is set, then the eviction attribute lru-memory-size maximum is overwritten with the value of local-max-memory. Both local-max-memory and lru-memory-size maximum are local member attributes and not cluster-wide. - You cannot configure lru-memory-size using gfsh.

   - 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.

> You can also configure Regions and Disk Stores using the gfsh command-line interface. See Region Commands and Disk Store Commands.

Related Topics
org.apache.geode.cache.RegionAttributes for data region persistence information
org.apache.geode.cache.EvictionAttributes for data region overflow information
org.apache.geode.cache.server.ClientSubscriptionConfig

Overflow Configuration Examples

This topic provides example of configurations for data overflow in VMware Tanzu GemFire.

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

This topic links to documentation that explains eviction in VMware Tanzu GemFire.

Use eviction to control data region size. Eviction actions are triggered by space-based thresholds.

- How Eviction Works

  Eviction settings cause VMware Tanzu 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.

- Configure Data Eviction

  Configure a region's eviction-attributes settings to keep your region within a specified limit.

How Eviction Works

This topic explains how eviction works in VMware Tanzu GemFire.

Eviction keeps a region's resource usage under a specified level by removing least recently used (LRU) entries to make way for new entries. You can choose whether expired entries are overflowed to disk or destroyed. See Persistence and Overflow.

Eviction is triggered when a size-based threshold is exceeded. A region's eviction threshold can be based on:

- entry count
- absolute memory usage

- percentage of available heap

These eviction algorithms are mutually exclusive; only one can be in effect for a given region.

When Tanzu 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 on the sizes of the older and newer entries.

For efficiency, the selection of items for removal is not strictly LRU, but does choose eviction candidates from among the region's oldest entries. As a result, eviction may leave older entries for the region in the local data store.

Eviction Actions

VMware Tanzu GemFire provides the following eviction actions:

- local destroy - Removes the entry from the local cache, but does not distribute the removal operation to remote members. This action can be applied to an entry in a partitioned region, but is not recommended if redundancy is enabled (redundant-copies > 0), as it introduces inconsistencies between the redundant buckets. When applied to an entry in a replicated region, Tanzu GemFire silently changes the region type to "preloaded" to accommodate the local modification.
- overflow to disk - The entry's value is overflowed to disk and set to null in memory. The entry's key is retained in the cache. This is the only eviction action fully supported for partitioned regions.

Eviction in Partitioned Regions

In partitioned regions, Tanzu GemFire removes the oldest entry in the bucket where the new entry operation is being performed. Tanzu GemFire maintains LRU entry information on a bucket-by-bucket basis, as the cost of maintaining information across the partitioned region would slow the system's performance.

- 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 exceeding 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.

Configure Data Eviction

This topic explains how to configure data eviction in VMware Tanzu GemFire.

Configure eviction to trigger based on criteria specific to an individual region, or the total application heap memory usage.

Configure the region eviction-attributes settings to keep that region within a specified limit.

Configure data eviction as follows. You do not need to perform these steps in the sequence shown.

Region-based eviction criteria (Entry count, bytes used)

1. Decide whether to evict based on:

    - Entry count. This is useful if your entry sizes are relatively uniform. Set this using eviction-entry-count.
    - Total bytes used.
        - In partitioned regions, set this using local-max-memory and eviction-max-memory, where the value you set is the same for each. If you set different values for these, the value for local-max-memory will override and replace the eviction-max-memory value.
        - In non-partitioned regions, set this using only the eviction-max-memory parameter.
2. Decide which of the following actions to take when the limit is reached:
    - Locally destroy the entry.
    - Overflow the entry data to disk. See Persistence and Overflow.
3. Decide the maximum amount of data to allow in the region 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. If using total bytes used by the region as your eviction criterion, decide whether to program a custom sizer for your region. If you can provide such a class, it might be faster than the standard sizing done by Tanzu GemFire. Your custom class must follow the guidelines for defining custom classes and, additionally, must implement org.apache.geode.cache.util.ObjectSizer. See Requirements for Using Custom Classes in Data Caching.

Application heap memory usage-based eviction

This eviction algorithm is based on the percentage of application heap used. This uses the Tanzu GemFire resource manager and is configured using eviction-heap-percentage. 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. Tanzu GemFire evicts the least recently used entry hosted by the member for each region. See Managing Heap and Off-heap Memory.

1. Decide the maximum amount of heap usage to be allowed in the member. The maximum amount of heap usage is defined on server startup as a percentage of total heap, and considers all heap memory usage from all sources, not just an individual region.
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.
3. Decide whether to program a custom sizer for your region. If you can provide such a class, it might be faster than the standard sizing done by Tanzu GemFire. Your custom class must follow the guidelines for defining custom classes and, additionally, must implement org.apache.geode.cache.util.ObjectSizer. See Requirements for Using Custom Classes in Data Caching.

Examples

Set an LRU memory eviction threshold of 1000 MB. Use a custom class for measuring the size of each object in the region:

```
gfsh>create region --name=myRegion --type=REPLICATE --eviction-max-memory=1000 \
--eviction-action=overflow-to-disk --eviction-object-sizer=com.myLib.MySizer
```

Create an eviction threshold on a partitioned region with a maximum entry count of 512:

```
gfsh>create region --name=myRegion --type=PARTITION --eviction-entry-count=512 \
--eviction-action=overflow-to-disk
```

To configure a partitioned region for heap LRU eviction, first configure the resource manager on server startup, then create a region with eviction enabled:

```
gfsh>start server --name=Server1 --eviction-heap-percentage=80
...
gfsh>create region --name=myRegion --type=PARTITION --eviction-action=overflow-to-disk
```

Expiration

This topic links to documentation that explains expiration in VMware Tanzu GemFire.

Use expiration to keep data current and to reduce region size by removing stale entries. Expiration actions are triggered by time-based thresholds.

- How Expiration Works

  Expiration removes old entries and entries that you are not using. You can choose whether expired entries are invalidated or destroyed.

- Configure Data Expiration

  Configure the type of expiration and the expiration action to use.

How Expiration Works

This topic explains how expiration works in VMware Tanzu GemFire.

Expiration keeps a region's data fresh by removing old entries and entries that you are not using. You can choose whether expired entries are invalidated or destroyed.

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 client/server system. The server (on the right) populates the region from a database and the data is automatically distributed throughout the system. The data is valid for only one hour, so the server performs a distributed destroy on entries that are an hour old. The client applications are consumers. The clients 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 clients will be forwarded to the server.

Expiration Types

VMware Tanzu GemFire provides two types of expiration, each triggered by a time-based threshold. These can co-exist; they are not mutually exclusive.

- 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. The attribute region-time-to-live is deprecated.
- 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. The attribute region-idle-time is deprecated.

: Querying does not update the time since the last access.

Expiration Actions

VMware Tanzu GemFire provides the following expiration actions:

- invalidate (default) - The data item's value is deleted, but the key remains in the cache. Applies to all distributed members in which the data item is replicated.
- destroy - The data item's key and value are both deleted. Applies to all distributed members in which the data item is replicated.
- local invalidate - Deletes the data item's value. Applies only to the local member.
- local destroy - Deletes the data item's key and value. Applies only to the local member.

You cannot use local-destroy or local-invalidate expiration actions in replicated or partitioned regions. You can use the local options only on distributed regions with a data-policy of empty, normal or preloaded.

Entry Expiration in Replicated Regions and Partitioned Regions

In replicated regions, entry updates are performed in the most convenient available copy of the data, then replicated to the other members, resetting their last-updated statistics to the same time. In partitioned regions, entry updates are always done in the primary copy, resetting the primary copy's last-updated and last-accessed statistics, then the secondary copies are updated to match.

In both replicated and partitioned regions, entry retrieval uses the most convenient available copy of the data, which may be any of the distributed copies. Retrievals are not propagated to other members. Differences in last-access times are reconciled when the data item is considered for expiration.

Expiration can be triggered in any copy of a replicated region, if the time elapsed since the last update or read access exceeds the established threshold. Expiration in partitioned regions is executed in the primary copy, based on the primary's last-accessed and last-updated statistics. In both cases, the expiration mechanism checks the last-accessed dates of all copies of the data item and updates the last-access date of all copies to the most recent last-accessed date. Then, if the elapsed time still puts the data item over the expiration threshold, the item is deleted in accordance with the expiration action specified for the 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 cluster. 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

You can configure VMware Tanzu GemFire to automatically remove or invalidate data after a certain amount of time. Expiration ensures that regions and entries do not retain stale data indefinitely. You can configure expiration by defining expiration attributes and actions that determine when and how data expires.

Types of Expiration

Expiration can apply at either the entry or region level. Each expiration type supports a specific set of configuration attributes and can be defined in either cache.xml or using gfsh.

Type	Applies To	Description	Resets On	cache.xml Definition	gfsh Command
Entry Time-to-Live (TTL)	Entry	Removes or invalidates entries after a fixed duration from creation or last update (depending on reset-policy).	Update (if reset-policy=last-modified)	entry-time-to-live	create region
Entry Idle Time	Entry	Removes or invalidates entries after a period of inactivity.	Read or Update	entry-idle-time	create region
Region Time-to-Live (TTL) Deprecated.	Region	Removes or invalidates all region data after a fixed duration.	Update	region-time-to-live	create region

Type	Applies To	Description	Resets On	cache.xml Definition	gfsh Command
Region Idle Time Deprecated.	Region	Removes or invalidates all region data after a period of inactivity.	Read or Update	region-idle-time	create region

Note: You can configure both Time-to-Live (TTL) and Idle Time for entries or regions. Expiration occurs when either condition is met first.

Enable Statistics

Expiration relies on region statistics to track access and modification times. You must enable statistics before expiration can take effect.

```
<region-attributes statistics-enabled="true">
```

You can enable statistics in any of the following ways:

- cache.xml - by setting the statistics-enabled attribute on the region.
- gfsh - add --enable-statistics=true when creating a region.
- Java API - call regionFactory.setStatisticsEnabled(true) before creating the region.

Configure Expiration Attributes

Once statistics are enabled, define expiration attributes to control how long data remains valid and what happens when entries or regions expire.

Available Attributes

Attribute	Applies To	Description
timeout	All	The duration (in seconds) before expiration occurs.
action	All	The behavior that occurs when data expires. Common actions include invalidate, destroy, local-invalidate, and local-destroy.
reset-policy	Entry TTL only	Controls whether TTL resets when an entry is modified. Values: never or last-modified (default).
custom-expiry	Entry-level only	Allows a custom Java class implementing CustomExpiry to determine per-entry expiration dynamically.

You can configure these attributes either in cache.xml or using the gfsh create region command.

Expiration Actions:

- Invalidate actions clear the value but keep the key in place.
- Destroy actions remove both key and value.
- Local actions apply only to the member where expiration occurs and do not propagate to others.

Important: The reset-policy attribute applies only to entry-time-to-live and is supported only on cache servers. It determines when the TTL timer resets:

- never: TTL starts when the entry is created.
- last-modified: TTL resets whenever the entry is updated (default).

Example (cache.xml)

```
<region-attributes statistics-enabled="true">
  <entry-time-to-live reset-policy="last-modified">
    <expiration-attributes timeout="60" action="local-invalidate" />
  </entry-time-to-live>
</region-attributes>
```

Example (gfsh)

```
create region --name=exampleRegion --type=REPLICATE \
  --enable-statistics=true \
  --entry-time-to-live-expiration=60 \
  --entry-time-to-live-action=local-invalidate \
  --entry-time-to-live-reset-policy=last-modified
```

Special Considerations

Distributed Expiration Behavior

Distributed expiration actions are coordinated between primary and secondary members when using distributed actions (invalidate, destroy).

- When a primary member determines an entry may have expired, it queries its secondaries for their most recent access timestamps.
- The primary adopts the most recent access time and reschedules expiration if the entry was recently accessed on another member.
- This coordination ensures consistent behavior across the cluster and prevents prematurely expiring entries that are still active elsewhere.
- Local actions (local-destroy, local-invalidate) are not distributed and do not use this coordination.

Partitioned Regions

- Only entry-level expiration is supported. Region-level attributes (region-time-to-live, region-idle-time) are ignored.
- For reliable read behavior, use entry-time-to-live instead of entry-idle-time.
- local-destroy and local-invalidate actions are not supported in partitioned regions.

Custom Expiration

You can override region-wide expiration for specific entries by providing a custom expiration class that implements org.apache.geode.cache.CustomExpiry.

This allows you to programmatically define entry-specific expiration times or actions.

Example Java Class

```
public class MyClass implements CustomExpiry, Declarable {
    private static final ExpirationAttributes CUSTOM =
        new ExpirationAttributes(10, ExpirationAction.DESTROY);

    @Override
    public ExpirationAttributes getExpiry(Entry entry) {
        int key = (Integer) entry.getKey();
        return key % 2 == 0 ? CUSTOM : null;
    }
}
```

Example (cache.xml)

```
<region-attributes statistics-enabled="true">
  <entry-idle-time>
    <expiration-attributes timeout="60">
      <custom-expiry>
        <class-name>com.company.mypackage.MyClass</class-name>
      </custom-expiry>
    </expiration-attributes>
  </entry-idle-time>
</region-attributes>
```

Example (gfsh)

```
create region --name=region1 --type=REPLICATE --enable-statistics \
  --entry-idle-time-expiration=60 \
  --entry-idle-time-custom-expiry=com.company.mypackage.MyClass
```

Performance Tuning for Expiration

By default, a single background thread handles all expiration events. If entries expire faster than that thread can process, it may lead to performance degradation or OutOfMemoryErrors.

To increase the number of threads used for expiration, set the following system property when starting the cache server:

```
-Dgemfire.EXPIRY_THREADS=<num_threads>
```

Increasing this value allows multiple threads to handle expiration concurrently.

Keeping the Cache Synchronized with Outside Data Sources

This topic links to documentation that explains how to keep your VMware Tanzu GemFire distributed cache synchronized with an outside data source by programming and installing application plug-ins for your region.

- Overview of Outside Data Sources

  Tanzu GemFire has application plug-ins to read data into the cache and write it from the cache.

- Configuring Database Connections Using JNDI.

  Use JNDI to maintain a connection pool that includes outside data sources.

- How Data Loaders Work

By default, regions have 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.

- Implement a Data Loader

  Program a data loader and configure your region to use the data loader.

Overview of Outside Data Sources

This topic provides an overview of the use of outside data sources with VMware Tanzu GemFire.

Tanzu 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 org.apache.geode.cache.CacheLoader. The CacheLoader.load method is called when the get operation cannot 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.
2. Write data out to the data source using the cache event handlers, CacheWriter and CacheListener. For implementation details, see Implementing Cache Event Handlers.
   - CacheWriter is run synchronously. Before performing any operation on a region entry, if any cache writers are defined for the region in the cluster, 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 cancel the region entry operation.
   - CacheListener is run synchronously 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 for more information.

Configuring Database Connections Using JNDI

This topic explains how to configure database connections for VMware Tanzu GemFire using JNDI.

To connect to external databases, for example 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 list shows DataSource connection types used in JTA transactions:

- XAPooledDataSource. Pooled SQL connections.
- ManagedDataSource. JNDI binding type for the J2EE Connector Architecture (JCA) ManagedConnectionFactory.
- 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, Tanzu GemFire logs a warning. For information about the DataSource interface, see: Interface DataSource on the Oracle documentation website.

Tanzu GemFire supports JDBC 2.0 and 3.0.

> Include any data source JAR files in your class path.

Example DataSource Configurations in cache.xml

The following sections show example cache.xml files configured for each of the DataSource connection types.

XAPooledDataSource cache.xml Example (Derby)

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 not encrypted.

When specifying the configuration properties for JCA-implemented database drivers that support XA transactions (in other words, XAPooledDataSource), you must use configuration properties to define the datasource connection instead of the connection-url attribute of the <jndi-binding> element. Configuration properties differ depending on your database vendor. Specify JNDI binding properties through the config-property tag, as shown in this example. You can add as many config-property tags as required.

```
<?xml version="1.0" encoding="UTF-8"?>
<cache
  xmlns="http://geode.apache.org/schema/cache"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cache-1.0.xsd"
  version="1.0"
  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="thecleartextpassword">
      <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>CreateDatabase</config-property-name>
        <config-property-type>java.lang.String</config-property-type>
        <config-property-value>create</config-property-value>
      </config-property>
      ...
    </jndi-binding>
  </jndi-bindings>
</cache>
```

JNDI Binding Configuration Properties for Different XAPooledDataSource Connections

The following are some example data source configurations for different databases. Consult your vendor database's documentation for additional details.

MySQL

```
...
<jndi-bindings>
  <jndi-binding type="XAPooledDataSource"
    ...
    xa-datasource-class="com.mysql.jdbc.jdbc2.optional.MysqlXADataSource">
    <config-property>
      <config-property-name>URL</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value>"jdbc:mysql://mysql-servername:3306/databasename"</config-property-value>
    </config-property>
    ...
  </jndi-binding>
  ...
</jndi-bindings>
```

PostgreSQL

```
...
<jndi-bindings>
  <jndi-binding type="XAPooledDataSource"
    ...
    xa-datasource-class="org.postgresql.xa.PGXADataSource">
    <config-property>
      <config-property-name>ServerName</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value>postgresql-hostname</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>postgresqldbname</config-property-value>
    </config-property>
    ...
  </jndi-binding>
  ...
</jndi-bindings>
```

Oracle

```
...
<jndi-bindings>
  <jndi-binding type="XAPooledDataSource"
    ...
    xa-datasource-class="oracle.jdbc.xa.client.OracleXADataSource">
    <config-property>
      <config-property-name>URL</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value>jdbc:oracle:oci8:@tc</config-property-value>
    </config-property>
    ...
  </jndi-binding>
  ...
</jndi-bindings>
```

Microsoft SQL Server

```
...
<jndi-bindings>
  <jndi-binding type="XAPooledDataSource"
    ...
    xa-datasource-class="com.microsoft.sqlserver.jdbc.SQLServerXADataSource">
    <config-property>
      <config-property-name>ServerName</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value>mysqlserver</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>databasename</config-property-value>
    </config-property>
    <config-property>
      <config-property-name>SelectMethod</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value>cursor</config-property-value>
    </config-property>
    ...
  </jndi-binding>
  ...
</jndi-bindings>
```

ManagedDataSource Connection Example (Derby)

ManagedDataSource connections for the JCA 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"?>
<cache xmlns="http://geode.apache.org/schema/cache"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cache-1.0.xsd"
  version="1.0"
  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="thecleartextpassword">
      <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>CreateDatabase</config-property-name>
        <config-property-type>java.lang.String</config-property-type>
        <config-property-value>create</config-property-value>
      </config-property>
      ...
    </jndi-binding>
  </jndi-bindings>
</cache>
```

PooledDataSource Example (Derby)

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 not encrypted.

```
<?xml version="1.0"?>
<cache xmlns="http://geode.apache.org/schema/cache"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cache-1.0.xsd"
  version="1.0"
  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="thecleartextpassword">
      <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>CreateDatabase</config-property-name>
        <config-property-type>java.lang.String</config-property-type>
        <config-property-value>create</config-property-value>
      </config-property>
      ...
    </jndi-binding>
  </jndi-bindings>
</cache>
```

SimpleDataSource Connection Example (Derby)

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"?>
<cache xmlns="http://geode.apache.org/schema/cache"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cache-1.0.xsd"
  version="1.0"
  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="thecleartextpassword"
      connection-url="jdbc:derby:newDB;create=true">
      ...
    </jndi-binding>
  </jndi-bindings>
</cache>
```

How Data Loaders Work

This topic explains how data loader work in VMware Tanzu GemFire.

By default, regions have 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 Tanzu GemFire cache from an outside data store. To do the reverse operation, writing data from the Tanzu GemFire cache to an outside data store, use a cache writer event handler. See Implementing Cache Event Handlers.

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 attributes 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 for information about configuring data sources.

> Partitioned regions generally require more JDBC connections than distributed regions.

Data source (database) connecting to cache loader on three Members

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 cluster can be running on different machines. Loading for the distributed region is performed from M1.

Data source (database) connecting to cache loader on Member 1. Cache locater on Member 1 connecting to Member 1, Member 2, and Member 3.

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

This topic explains how to use a data loader with VMware Tanzu GemFire.

To use a data loader:

1. Implement the org.apache.geode.cache.CacheLoader interface.
2. Configure and deploy the implementation.

Implement the CacheLoader Interface

For a get operation, if the key is not in the cache, the thread serving the get operation invokes the CacheLoader.load method. Implement load to return the value for the key, which will be placed into the region in addition to being returned to the caller.

org.apache.geode.cache.CacheLoader inherits from Declarable, so implement the Declarable.initialize method if your CacheLoader implementation needs to be initialized with some arguments. Specify the required arguments either in your cache.xml file or in a gfsh create region or alter region command. Do not define the Declarable.init() method; it is deprecated.

Here is an example implementation:

```
public class SimpleCacheLoader implements CacheLoader {
  public Object load(LoaderHelper helper) {
    String key = (String) helper.getKey();

    // Return an entry value created from the key, assuming that
    // all keys are of the form "key1", "key2", "keyN"
    return "LoadedValue" + key.substring(3);
  }
}
```

If you need to run Region API calls from your implementation, spawn separate threads for them. Do not make direct calls to Region methods from your load method, as it could cause the cache loader to block, hurting the performance of the cluster.

Configure and Deploy

Use one of these three ways to configure and deploy the cache loader:

Option 1: If configuring a cluster by defining a cache.xml file, deploy by adding the cache loader to the class path when starting servers.

Here is an example configuration within the cache.xml file that specifies the loader without arguments:

```
<region-attributes>
   <cache-loader>
      <class-name>myLoader</class-name>
   </cache-loader>
</region-attributes>
```

Or, here is an example configuration within the cache.xml file that specifies the loader with an argument:

```
<cache-loader>
   <class-name>com.company.data.DatabaseLoader</class-name>
   <parameter name="URL">
      <string>jdbc:cloudscape:rmi:MyData</string>
   </parameter>
</cache-loader>
```

Add the cache loader JAR file to the class path when starting servers. For example:

```
gfsh>start server --name=s2 --classpath=/var/data/lib/myLoader.jar
```

Option 2: If deploying the JAR file at server startup, add the JAR file to the class path and use gfsh to apply the configuration to the region.

To deploy the JAR file, add the cache loader JAR file to the class path when starting servers. For example:

```
gfsh>start server --name=s2 --classpath=/var/data/lib/myLoader.jar
```

Use gfsh to apply the configuration of the CacheLoader implementation to the region with gfsh create region or gfsh alter region. Here is an example of region creation without arguments:

```
gfsh>create region --name=r3 --cache-loader=com.example.appname.myCacheLoader
```

Here is an example of region creation with an argument:

```
gfsh>create region --name=r3 \
--cache-loader=com.example.appname.myCacheLoader{'URL':'jdbc:cloudscape:rmi:MyData'}
```

Here is an example of altering a region:

```
gfsh>alter region --name=r3 --cache-loader=com.example.appname.myCacheLoader
```

Option 3 applies to partitioned regions: If deploying the JAR file with the gfsh deploy command after servers have been started, use gfsh to apply the configuration to the region.

After server creation use gfsh to deploy the JAR file to all the servers. For example:

```
gfsh>deploy --jars=/var/data/lib/myLoader.jar
```

We do not generally use the gfsh deploy command when the servers host replicated regions, as detailed in How Data Loaders Work.

Use gfsh to apply the configuration of the CacheLoader implementation to the region with gfsh create region or gfsh alter region. Here is an example of region creation without arguments:

```
gfsh>create region --name=r3 --cache-loader=com.example.appname.myCacheLoader
```

Here is an example of region creation with an argument:

```
gfsh>create region --name=r3 \
--cache-loader=com.example.appname.myCacheLoader{'URL':'jdbc:cloudscape:rmi:MyData'}
```

Here is an example of altering a region:

```
gfsh>alter region --name=r3 --cache-loader=com.example.appname.myCacheLoader
```

Implementing a Server or Peer with a Cache Loader

Servers and peers with an embedded cache can configure a cache loader in only the members where it makes sense to do so. The design 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.

Implement the org.apache.geode.cache.CacheLoader interface. Region creation configures the cache loader as in this example:

```
RegionFactory<String,Object> rf = cache.createRegionFactory(REPLICATE);
rf.setCacheLoader(new QuoteLoader());
quotes = rf.create("NASDAQ-Quotes");
```

Data Serialization

This topic provides links to documentation about data serialization in VMware Tanzu GemFire.

Data that you manage in Tanzu 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

  Tanzu GemFire offers serialization options other than Java serialization that give you higher performance and greater flexibility for data storage, transfers, and language types.

- Tanzu GemFire PDX Serialization

  Tanzu 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.

- Tanzu GemFire Data Serialization (DataSerializable and DataSerializer)

  Tanzu GemFire's DataSerializable interface gives you quick serialization of your objects.

- 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.

Overview of Data Serialization

This topic provides an overview of VMware Tanzu GemFire data serialization options.

Tanzu 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 Tanzu GemFire moves out of the local cache must be serializable. However, you do not necessarily need to implement java.io.Serializable because other serialization options are available in Tanzu 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

> If you are storing objects with the Session Management Extensions, these objects must be serializable because they are serialized before being stored in the region.

To minimize the cost of serialization and deserialization, Tanzu 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 initially stored in serialized form.

Data Serialization Options

With Tanzu GemFire, you have the option to serialize your domain objects automatically or to implement serialization using one of Tanzu 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 Tanzu 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 Tanzu GemFire cache configuration.

Tanzu 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.

Serialization Options: Comparison of Features

Capability	Tanzu GemFire Data Serializable	Tanzu GemFire PDX Serializable
Implements Java Serializable.	X	

Capability	Tanzu GemFire Data Serializable	Tanzu GemFire PDX Serializable
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 Tanzu GemFire		X
Works with .NET clients.	X	X
Works with C++ clients.	X	X
Works with Tanzu GemFire delta propagation.	X	X (See note below.)

> By default, you can use Tanzu GemFire delta propagation with PDX serialization. However, delta propagation will not work if you have set the Tanzu GemFire property read-serialized to “true”. In terms of deserialization, to apply a change delta propagation requires a domain class instance and the fromDeltamethod. 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 Tanzu GemFire Serialization (PDX or Data Serializable) and Java Serialization

Tanzu GemFire serialization (either PDX Serialization or Data Serialization) does not support circular object graphs whereas Java serialization does. In Tanzu 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.

GemFire PDX Serialization

This topic discusses using PDX serialization in VMware Tanzu GemFire.

Tanzu 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.

- Tanzu GemFire PDX Serialization Features

  Tanzu GemFire PDX serialization offers several advantages in terms of functionality.

- High Level Steps for Using PDX Serialization

  To use PDX serialization, you can configure and use Tanzu GemFire’s reflection-based autoserializer, or you can program the serialization of your objects by using the PDX interfaces and classes.

- 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.

- 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.

- 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.

- 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.

- Adding JSON Documents to the Tanzu 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.

- 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.

- Persisting PDX Metadata to Disk

  Tanzu GemFire allows you to persist PDX metadata to disk and specify the disk store to use.

- Using PDX Objects as Region Entry Keys

  Using PDX objects as region entry keys is highly discouraged.

GemFire PDX Serialization Features

This topic lists VMware Tanzu GemFire PDX serialization features. Tanzu 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 cluster if the versions differ by the addition or removal of fields. This compatibility lets you gradually introduce modified code and data into the cluster, without bringing the cluster down.

Tanzu GemFire maintains a central registry of the PDX domain object metadata. Using the registry, Tanzu 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, Tanzu 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, Tanzu GemFire stores the object's type information in the central registry. The information is passed among clients and servers, peers, and clusters.

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.

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 other 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

This topic explains, at a high level, how to use PDX serialization in VMware Tanzu GemFire.

To use PDX serialization, you can configure and use Tanzu 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
    - Serializing Your Domain Object with a PdxSerializer
    - Implementing PdxSerializable in Your Domain Object
2. To ensure that your servers do not need to load the application classes, set the pdx read-serialized attribute to true. In gfsh, execute the following command before starting up your servers:

    ```
    gfsh>configure pdx --read-serialized=true
    ```

    By using gfsh, this configuration can be propagated across the cluster through the Cluster Configuration Service. Alternately, you would need to configure pdx read-serialized in each server's cache.xml file.
3. If you are storing any Tanzu GemFire data on disk, then you must configure PDX serialization to use persistence. See Persisting PDX Metadata to Disk for more information.
4. (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.

PDX and Multi-Site (WAN) Deployments

For multisite (WAN) installations only: If you will use PDX serialization in any of your WAN-enabled regions, for each cluster, 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.

PDX Serialization Best Practices

This topic describes recommended practices for using PDX serialization in Tanzu GemFire deployments.

Field Compatibility

- Add new fields or types as needed.
- Do not change the data type of an existing field, for example, avoid changing a customer ID from a number to a string.
- Do not change the order of values in an enum.

PDX Type Persistence

- Persist PDX types to a disk store on the server.
- Use a dedicated disk store specifically for PDX types.

Backup and Recovery

- Regularly back up PDX-type disk stores.
    - Perform both full and incremental backups.
    - Take periodic snapshot exports for recovery.
- The PDX type disk store is part of the overall GemFire backup. Use GemFire's built-in backup mechanism to ensure data consistency. For more information, see Creating Backups for System Recovery and Operational Management.

Client-Side PDX Handling

- Clear PDX type IDs on client disconnect to avoid conflicts.
    - Add the following JVM argument on the client:

```
-J-Dgemfire.ON_DISCONNECT_CLEAR_PDXTYPEIDS=true
```

WAN Configuration

- Assign a unique distributed-system-id for each WAN site.
- Do not reuse a distributed-system-id across sites.

Using Automatic Reflection-Based PDX Serialization

This topic explains how to use automatic reflection-based PDX serialization in VMware Tanzu GemFire.

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 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 for details.

> Your custom PDX autoserializable classes cannot use the org.apache.geode package. If they do, the classes will be ignored by the PDX auto serializer.

Prerequisites

- Understand generally how to configure the Tanzu 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. In the domain classes that you wish to autoserialize, make sure each class has a zero-arg constructor. For example:

   ```
   public PortfolioPdx(){}
   ```

2. Using one of the following methods, set the PDX serializer to ReflectionBasedAutoSerializer.

   1. In gfsh, execute the following command prior to starting up any members that host data:

      ```
      gfsh>configure pdx --auto-serializable-classes=com\.company\.domain\..*
      ```

      By using gfsh, this configuration can propagated across the cluster through the Cluster Configuration Service.

   2. Alternately, in cache.xml:

      ```
      <!-- Cache configuration configuring auto serialization behavior -->
      <cache>
        <pdx>
          <pdx-serializer>
            <class-name>
              org.apache.geode.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.

3. Using the Java API:

```
Cache c = new CacheFactory()
  .setPdxSerializer(new ReflectionBasedAutoSerializer("com.company.domain.DomainObject"))
  .create();
```

3. 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 for details.
   - By creating a subclass of ReflectionBasedAutoSerializer and overriding specific methods. See Extending the ReflectionBasedAutoSerializer for details.
4. 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 gfsh, use the following command:

     ```
     gfsh>configure pdx --portable-auto-serializable-classes=com\.company\.domain\..*
     ```

     By using gfsh, this configuration can propagated across the cluster through the Cluster Configuration Service.
   - In cache.xml:

     ```
     <!-- Cache configuration configuring auto serialization behavior -->
     <cache>
       <pdx>
         <pdx-serializer>
           <class-name>
             org.apache.geode.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.

> 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

This topic explains how to customize serialization with class patter strings in VMware Tanzu GemFire.

Use class pattern strings to name the classes that you want to serialize using Tanzu 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
```

> 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
```

> 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 VMware Tanzu GemFire ReflectionBasedAutoSerializer to handle serialization in a customized manner. This topic 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 Tanzu 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, 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 the JavaDocs on ReflectionBasedAutoSerializer for details.

Example of Optimizing Autoserialization 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 autoserialization:

```
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

This topic explains how to serialize a domain object with a PdxSerializer in VMware Tanzu GemFire.

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.

> The PdxSerializer toData and fromData methods differ from those for PdxSerializable. They have different parameters and results.

Procedure

1. In the domain classes that you wish to PDX serialize, make sure each class has a zero-arg constructor. For example:

   ```
   public PortfolioPdx(){}
   ```

2. If you have not already implemented PdxSerializer for some other domain object, perform these steps:

   1. 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 org.apache.geode.cache.Declarable;
      import org.apache.geode.pdx.PdxReader;
      import org.apache.geode.pdx.PdxSerializer;
      import org.apache.geode.pdx.PdxWriter;

      public class ExamplePdxSerializer implements PdxSerializer, Declarable {
      ...
      ```

   2. 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();
```

> You cannot specify a custom pdx-serializer class using gfsh, however the configure pdx command automatically configures the org.apache.geode.pdx.ReflectionBasedAutoSerializer class. See configure pdx.

3. Program PdxSerializer.toData to recognize, cast, and handle your domain object:
    1. Write each standard Java data field of your domain class using the PdxWriter write methods.
    2. Call the PdxWriter markIdentityField method for each field you want to have Tanzu GemFire use to identify your object. Put this after the field's write method. Tanzu GemFire uses this information to compare objects for operations like distinct queries. If you do not set as 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.
    3. 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.
    4. For best performance, do fixed width fields first and then variable length fields.
    5. Optionally, 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;
 }
```

6. 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.

   Tanzu 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;
}
```

4. Optionally, you can enable extra validation in your use of PdxWriter. To enable this, set the system property gemfire.validatePdxWriters to true. VMware recommends that you only set this option if you are debugging new code because this option can decrease system performance.

Implementing PdxSerializable in Your Domain Object

This topic explains how to implement PdxSerializable in your domain object in VMware Tanzu GemFire.

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 org.apache.geode.pdx classes.

   For example:

   ```
   import org.apache.geode.pdx.PdxReader;
   import org.apache.geode.pdx.PdxSerializable;
   import org.apache.geode.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.

   1. Write each standard Java data field of your domain class using the PdxWriter write methods. Tanzu GemFire automatically provides PdxWriter to the toData method for PdxSerializable objects.
   2. Call the PdxWriter markIdentifyField method for each field you want to have Tanzu GemFire use to identify your object. Put this after the field's write method. Tanzu GemFire uses this information to compare objects for operations like distinct queries. If you do not set as 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.
   3. 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.
   4. 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 markIdentifyField 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.

   Tanzu 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 Tanzu GemFire applications to use PdxInstance for selective object deserialization. See Programming Your Application to Use PdxInstances.

Programming Your Application to Use PdxInstances

This topic explains how to program applications to use a PdxInstance in VMware Tanzu GemFire.

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.

> 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.

> 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.

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 as least one identity field, then the PdxInstanceequals 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 Tanzu GemFire cache. See Basic Configuration and Programming.

Procedure

In your application where you fetch data from the cache, provide the following configuration and code as appropriate:

1. In the cache.xml file of the member where entry fetches are run, set the <pdx> read-serialized attribute to true. 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>
```

2. 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)myPdxInstance.getObject();
}
else if (myObject instanceof DomainClass) {
  // get returned instance of domain object
  // code to handle domain object instance
  ...
}
...
```

> 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

This topic discusses the use of the JsonDocumentFactory API in VMware Tanzu GemFire. Previous versions of Tanzu GemFire use the JSONFormatter API.

The JSONFormatter API has been deprecated in Tanzu GemFire v10. For information about the JSONFormatter API, see Adding JSON Documents to the VMware Tanzu GemFire Cache in the Tanzu GemFire v9.15 documentation.

> Caution: Do not use JsonDocuments using the BSON format or KeepSerialized unless all members have been upgraded to Tanzu GemFire v10. Attempts to use these before all members have been upgraded to Tanzu GemFire v10 will result in a deserialization exception on the client.

Overview

The JsonDocumentFactory API allows you to convert a JSON String into a binary form that can be stored in a Tanzu GemFire region. The JSON String must be a single JSON object, starting and ending with curly braces. Tanzu GemFire accesses each field in the binary form without needing to re-parse JSON or deserialize the binary form.

Usage

For querying and indexing, any field contained inside a JsonDocument can be used, including any nested field within JSON objects or JSON arrays.

- To specify a field of a JSON object for the query or index, specify the field name. In some cases a dot before the field name may be required.
- To specify an element of a JSON array, use brackets. For example, jsonArray[1].

When an application gets a JsonDocument, the application can convert it back to a JSON String by calling toJson(), or the application can access any of its fields by calling getField(fieldName). For a full list of methods that can be used to read the fields, see the Document interface.

Tanzu GemFire v10 supports two different binary formats:

- BSON: Default. Based on the BSON standard. Recommended if your data does not have a well defined schema.
- PDX: Recommended for data that has a well-defined schema.

PDX stores the schema, which is beneficial if the schema is reused often. If the schema must be created for each document, PDX becomes more costly than BSON.

REST clients can both access and store JsonDocuments. The REST client stores JsonDocuments using the PDX format, but they can access JsonDocuments that are stored using the BSON format.

For more information about these APIs, see the Java API documentation for the org.apache.geode.json package.

BSON

- Get a JsonDocumentFactory: To get a JsonDocumentFactory that uses the BSON format, call getJsonDocumentFactory() on your Tanzu GemFire Cache. The interface that represents the binary form is JsonDocument.

  > Caution: Do not use JsonDocuments using the BSON format or KeepSerialized unless all members have been upgraded to Tanzu GemFire v10. Attempts to use these before all members have been upgraded to Tanzu GemFire v10 will result in a deserialization exception on the client.

- Access By Clients Running Earlier Versions of GemFire: JsonDocuments using the BSON format cannot be accessed in a client running earlier versions of Tanzu GemFire. This is also true of the

KeepSerialized interface. Attempts to do so will result in a deserialization exception on the client. JsonDocuments using the BSON format cannot be sent to older versions of Tanzu GemFire in a cluster or over WAN during rolling upgrade. This is also true of the KeepSerialized interface. The BSON format and the KeepSerialized interface should only be used once all the members have been upgraded to Tanzu GemFire v10.

- Queries on Regions: When performing queries on regions with JsonDocuments that use the BSON format, query results that contain one or more instances of org.apache.geode.cache.query.Struct will always show a StructType of “Object”.

PDX

- Get a JsonDocumentFactory: To get a factory that uses PDX call getJsonDocumentFactory(StorageFormat.PDX) on your Tanzu GemFire Cache.
- Access By Clients Running Earlier Versions of GemFire: JsonDocuments using the PDX format can be accessed by clients running earlier versions of Tanzu GemFire.
- Java Domain Class Mapping: JsonDocuments using the PDX format have support for mapping the document to a Java domain class using a “@type” field in the document.
- Queries on Regions: When performing on regions with JsonDocuments that use the PDX format, query results that contain one of more instances of org.apache.geode.cache.query.Struct will include more specific StructType values like “String” and “Integer”.

Using PdxInstanceFactory to Create PdxInstances

This topic discusses using the PdxInstanceFactory to create a PdxInstances in VMware Tanzu GemFire. 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 as least one identity field using the markIndentityField method. If you do not mark an identity field, the PdxInstanceequals 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. An 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 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

This topic explains how to use VMware Tanzu GemFire to persist PDX metadata to disk and specify the disk store to use.

Prerequisites

- Understand generally how to configure the Tanzu GemFire cache. See Basic Configuration and Programming.
- Understand how Tanzu GemFire disk stores work. See Disk Storage.

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 Tanzu GemFire default disk store, set the <pdx> attribute disk-store-name to the name of your non-default disk store.

   > 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.

3. (Optional) If you later want to rename the PDX types that are persisted to disk, you can do so on your offline disk-stores by executing the pdx rename command. See pdx rename.

Example cache.xml:

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
  ...
```

Using PDX Objects as Region Entry Keys

This topic discusses using PDX objects as region entry keys in VMware Tanzu GemFire.

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.

GemFire Data Serialization (DataSerializable and DataSerializer)

This topic discusses using VMware Tanzu GemFire's DataSerializable interface for quick serialization of your objects.

Data Serialization with the DataSerializable Interface

Tanzu GemFire's DataSerializable interface gives you faster and more compact data serialization than the standard Java serialization or Tanzu GemFire PDX serialization. However, while Tanzu GemFire DataSerializable interface is generally more performant than Tanzu 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 Instantiator is by specifying it in the serialization-registration element of cache.xml.

For more information, see the online Java documentation for DataSerializable and DataSerializer.

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>
```

In addition to speeding standard object serialization, you can use the DataSerializable interface to serialize any custom objects you store in the cache.

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 JavaDocs on DataSerializable and DataSerializer for more information.

Standard Java Serialization

This topic discusses using standard Java serialization in VMware Tanzu GemFire.

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.

Events and Event Handling

This topic links to documentation about events and event handling in VVMware Tanzu GemFire.

Tanzu GemFire provides versatile and reliable event distribution and handling for your cached data and system member events.

- How Events Work

  Members in your cluster receive cache updates from other members through cache events. The other members can be peers to the member, clients or servers or other clusters.

- Implementing Tanzu GemFire Event Handlers

  You can specify event handlers for region and region entry operations and for administrative events.

- Configuring Peer-to-Peer Event Messaging

  You can receive events from cluster peers for any region that is not a local region. Local regions receive only local cache events.

- Configuring Client/Server Event Messaging

  You can receive events from your servers for server-side cache events and query result changes.

- Configuring Multi-Site (WAN) Event Queues

  In a multi-site (WAN) installation, Tanzu 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.

How Events Work

This topic describes how events work in VMware Tanzu GemFire.

Members in your Tanzu GemFire cluster receive cache updates from other members through cache events. The other members can be peers to the member, clients or servers or other clusters.

Events Features

These are the primary features of Tanzu 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.

> 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 Tanzu 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.

    > 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.

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

This topic explains peer-to-peer event distribution in VMware Tanzu GemFire.

When a region or entry operation is performed, Tanzu GemFire distributes the associated events in the cluster 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, Tanzu GemFire guarantees ordering of events across threads, but for distributed regions it does not. 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

This topic explains client-to-server event distribution in VMware Tanzu GemFire.

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.

Client entry update propagation as described below.

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 cancel 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

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.

Event subscription event distribution as described in this section.

Change in Server Cache	Effect on Client Cache
Entry create/update	For subscriptions with receiveValues set to true, entry create or update. For subscriptions with receiveValues 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.

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 cluster. 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.

Server-to-client message tracking example as described.

The client's message tracking list holds the highest sequence ID of any message received for each originating thread. The list can become excessively 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 Tanzu 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.

> 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.
>
> 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

This topic explains multi-site (WAN) event distribution in VMware Tanzu GemFire.

Tanzu GemFire distributes a subset of cache events between clusters, 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. For serial gateway queues, 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 enable-persistence 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 cluster 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

This topic lists VMware Tanzu GemFire event handlers and events.

Tanzu GemFire provides many types of events and event handlers to help you manage your data and application needs.

Event Handlers

Use either cache handlers or membership 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 Implementing an AsyncEventListener for Write-Behind Cache Event Handling.
CacheCallback		Superinterface of all cache event listeners. Functions only to clean up resources that the callback allocated.
CacheListener	RegionEvent, EntryEvent	Tracks changes to region and its data entries. Responds synchronously. Extends CacheCallback 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 *pending* changes to the region and its data entries in this member or one of its peers. Has the ability to cancel the operations in question. Extends CacheCallback 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.
ClientMembershipListener	ClientMembershipEvent	One of the interfaces that replaces the deprecated Admin APIs. You can use the ClientMembershipListener to receive membership events only about clients. This listener's callback methods are invoked when this process detects connection changes to clients. Callback methods include memberCrashed, memberJoined, memberLeft (graceful exit).
CqListener	CqEvent	Receives events from the server cache that satisfy a client-specified query. Extends CacheCallback interface. Installed in the client inside a CqQuery.
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.
MembershipListener	MembershipEvent	Use this interface to receive membership events only about peers. This listener's callback methods are invoked when peer members join or leave the cluster. Callback methods include memberCrashed, memberJoined, and memberLeft (graceful exit).

Handler API	Events received	Description
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 cluster. Extends CacheCallback and CacheListener. Installed in region as a CacheListener.
TransactionListener	TransactionEvent with embedded list of EntryEvent	Tracks the outcome of transactions and changes to data entries in the transaction. 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 *pending* transaction commits. Has the ability to cancel the transaction. Extends CacheCallback interface. Installed in cache using transaction manager. At most one writer is called per transaction. Install a writer in every transaction host.
UniversalMembershipListenerAdapter	MembershipEvent and ClientMembershipEvent	One of the interfaces that replaces the deprecated Admin APIs. Provides a wrapper for MembershipListener and ClientMembershipListener callbacks for both clients and peers.

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.
CacheEvent		Superinterface to RegionEvent and EntryEvent. 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.
ClientMembershipEvent	ClientMembershipListener	An event delivered to a ClientMembershipListener when this process detects connection changes to servers or clients.
CqEvent	CqListener	Provides information about a change to the results of a continuous query running on a server on behalf of a client. CqEvents are processed on the client.

Event	Passed to handler ...	Description
EntryEvent	CacheListener, CacheWriter, TransactionListener (inside the TransactionEvent)	Extends CacheEvent 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. EntryEvent.getNewValue returns the current value of the data entry. EntryEvent.getOldValue 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. EntryEvent provides the Tanzu GemFire transaction ID if available. You can retrieve serialized values from EntryEvent using the getSerialized* 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 put function as the put recognizes that the value is serialized and bypasses the serialization step.
MembershipEvent (membership event)	MembershipListener	An event that describes the member that originated this event. Instances of this are delivered to a MembershipListener when a member has joined or left the cluster.
RegionEvent	CacheListener, CacheWriter, RegionMembershipListener	Extends CacheEvent for region events. Provides information about operations that affect the whole region, such as reinitialization of the region after being destroyed.
TimestampedEntryEvent	GatewayConflictResolver	Extends EntryEvent 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.
TransactionEvent	TransactionListener, TransactionWriter	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 EntryEvent instances. The entry events are listed in the order in which the operations were performed in the transaction. 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.

Implementing GemFire Event Handlers

This topic links to documentation about implementing VMware Tanzu 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.

- Implementing an AsyncEventListener for Write-Behind Cache Event Handling

  An AsyncEventListener asynchronously processes batches of events after they have been applied to a region. You can use an AsyncEventListener implementation as a write-behind cache event

handler to synchronize region updates with a database.

- 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.

- Cache Event Handler Examples

  Some examples of cache event handlers.

Implementing Cache Event Handlers

This topic explains how to implement cache event handlers in VMware Tanzu GemFire.

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, Tanzu GemFire provides a convenience class with empty stubs for the interface callback methods.

> 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.

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 org.apache.geode.cache.util show the options.
3. Program each event handler:
   1. Extend the handler's adapter class.
   2. If you want to declare the handler in the cache.xml, implement the org.apache.geode.cache.Declarable interface as well.
   3. Implement the handler's callback methods as needed by your application.

      > 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.

      Example:

```
package myPackage;
import org.apache.geode.cache.Declarable;
import org.apache.geode.cache.EntryEvent;
import org.apache.geode.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:

```
tradesRegion = cache.createRegionFactory(RegionShortcut.PARTITION)
  .addCacheListener(new MyCacheListener())
  .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:

```
CacheListener listener1 = new myCacheListener1();
CacheListener listener2 = new myCacheListener2();
CacheListener listener3 = new myCacheListener3();

Region nr = cache.createRegionFactory()
  .initCacheListeners(new CacheListener[]
    {listener1, listener2, listener3})
  .setScope(Scope.DISTRIBUTED_NO_ACK)
  .create(name);
```

Implementing an AsyncEventListener for Write-Behind Cache Event Handling

This topic explains how to implement an AsyncEventListener in VMware Tanzu GemFire for write-behind cache event handling.

An AsyncEventListener asynchronously processes batches of events after they have been applied to a region. 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 one or more threads dispatch batches 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 Tanzu GemFire member, and it delivers all of a region's events, in order of occurrence, to a configured AsyncEventListener implementation. A parallel queue is deployed to multiple Tanzu GemFire members, and each instance of the queue delivers region events, possibly simultaneously, 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 Tanzu 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 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 Tanzu 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. When you configure multiple threads for a parallel queue, each queue hosted on a Tanzu GemFire member is processed by dispatcher threads; the total number of queues created depends on the number of members that host the region.

A GatewayEventFilter can be placed on the AsyncEventQueue to control whether a particular event is sent to a selected AsyncEventListener. For example, events associated with sensitive data could be detected and not queued. For more detail, see the Javadocs for GatewayEventFilter.

A GatewayEventSubstitutionFilter can specify whether the event is transmitted in its entirety or in an altered representation. For example, to reduce the size of the data being serialized, it might be a more efficient to represent a full object by only its key. For more detail, see the Javadocs for GatewayEventSubstitutionFilter.

Operation Distribution from an AsyncEventQueue

An AsyncEventQueue distributes these operations:

- Entry create
- Entry put
- Entry distributed destroy, providing the operation is not an expiration action
- Expiration destroy, if the forward-expiration-destroy attribute is set to true. By default, this attribute is false, but you can set it to true using cache.xml or gfsh. To set this attribute in the Java API, use

AsyncEventQueueFactory.setForwardExpirationDestroy(). See the javadocs for details.

These operations are not distributed:

- Get
- Invalidate
- Local destroy
- Region operations
- Expiration actions
- Expiration destroy, if the forward-expiration-destroy attribute is set to false. The default value is false.

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 within a thread when delivering events to your listener implementation. Use parallel queues when the order of events within a thread is not important, and when you require maximum throughput for processing events. In both cases, serial and parallel, the order of operations on a given key is preserved within the scope of the thread.
- You must install the AsyncEventListener implementation on a Tanzu GemFire member that hosts the region whose events you want to process.
- If you configure a parallel AsyncEventQueue, deploy the queue on each Tanzu 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 listener for dispatching events. The listeners on other members remain on standby for redundancy. For best performance and most efficient use of memory, install only one standby listener (redundancy of at most one).
- 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 Tanzu 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 boolean processEvents(List<AsyncEvent> events) {

    // Process each AsyncEvent

    for(AsyncEvent event: events) {

      // Write the event to a database

    }
  }
}
```

Processing AsyncEvents

Use the AsyncEventListener.processEvents method to process AsyncEvents. This method is called asynchronously when events are queued to be processed. The size of the list reflects the number of batch events where batch size is defined in the AsyncEventQueueFactory. The processEvents method returns a boolean; true if the AsyncEvents are processed correctly, and false if any events fail processing. As long as processEvents returns false, Tanzu GemFire continues to re-try processing the events.

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<>();

  List<JdbcBatch> updatedEntries = new ArrayList<>();
  List<String> destroyedEntries = new ArrayList<>();
  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

1. 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 WAN Configuration for more information.

   gfsh configuration

   ```
   gfsh>create async-event-queue --id=sampleQueue --persistent --disk-store=exampleStore --listener=com.myCompan
   y.MyAsyncEventListener --listener-param=url#jdbc:db2:SAMPLE,username#gfeadmin,password#admin1
   ```

   The parameters for this command uses the following syntax:

   ```
   create async-event-queue --id=value --listener=value [--group=value] [--batch-size=value]
   [--persistent(=value)?] [--disk-store=value] [--max-queue-memory=value] [--listener-param=value(,value)*]
   ```

   For more information, see create async-event-queue.

   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("sampleQueue", listener);
   ```

2. If you are using a parallel AsyncEventQueue, the gfsh example above requires no alteration, as gfsh applies to all members. If using cache.xml or the Java API to configure your AsyncEventQueue, repeat the above configuration in each Tanzu GemFire member that will host the region. Use the same ID and configuration settings for each queue configuration.

> You can ensure other members use the sample configuration by using the cluster configuration service available in gfsh. See Overview of the Cluster Configuration Service.

3. On each Tanzu GemFire member that hosts the AsyncEventQueue, assign the queue to each region that you want to use with the AsyncEventListener implementation.

   gfsh Configuration

   ```
   gfsh>create region --name=Customer --async-event-queue-id=sampleQueue
   ```

   Note that you can specify multiple queues on the command line in a comma-delimited list.

   cache.xml Configuration

   ```
   <cache>
     <region name="Customer">
       <region-attributes async-event-queue-ids="sampleQueue">
       </region-attributes>
     </region>
     ...
   </cache>
   ```

   Java Configuration

   ```
   RegionFactory rf1 = cache.createRegionFactory();
   rf1.addAsyncEventQueue(sampleQueue);
   Region customer = rf1.create("Customer");

   // Assign the queue to multiple regions as needed
   RegionFactory rf2 = cache.createRegionFactory();
   rf2.addAsyncEventQueue(sampleQueue);
   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("sampleQueue");
   ```

   See AttributesMutator in the Tanzu GemFire API documentation for more information.

4. Optionally configure persistence and conflation for the queue.

> You must configure your AsyncEventQueue to be persistent if you are using persistent data regions. Using a non-persistent queue with a persistent region is not supported.

> You can configure multiple regions to share the same AsyncEventQueue if they use the same AsyncEventListener. The maximum combined number of AsyncEventQueues and GatewaySenders per cluster is 128.

5. 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.

The AsyncEventListener receives events from every region configured with the associated AsyncEventQueue.

How to Safely Modify the Cache from an Event Handler Callback

This topic explains how to safely modify the cache from an event handler callback in VMware Tanzu GemFire.

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. Tanzu 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 Tanzu GemFire DistributedLockService.
- Modifying region attributes.
- Executing a function through the Tanzu GemFire FunctionService.

To be on the safe side, do not make any calls to the Tanzu GemFire API directly from your event handler. Make all Tanzu GemFire API calls from within a separate thread or executor.

How to Perform Distributed Operations Based on Events

If you need to use the Tanzu 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.

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 (org.apache.geode.cache.RegionDestroyedException e) {
        ...
      }
      catch (org.apache.geode.cache.EntryExistsException e) {
        ...
      }
    }
  });
}
```

For additional information about the Executor, see the SerialExecutor example on the Oracle Java web site.

Cache Event Handler Examples

This topic provides examples of cache event handlers in VMware Tanzu GemFire.

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 the RegionFactory method addCacheListener.

```
Region newReg = cache.createRegionFactory()
  .addCacheListener(new SimpleCacheListener())
  .create(name);
```

You can create a cache writer similarly, using the RegionFactory method setCacheWriter, like this:

```
Region newReg = cache.createRegionFactory()
  .setCacheWriter(new SimpleCacheWriter())
  .create(name);
```

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:

```
CacheListener listener1 = new myCacheListener1();
CacheListener listener2 = new myCacheListener2();
CacheListener listener3 = new myCacheListener3();

Region nr = cache.createRegionFactory()
  .initCacheListeners(new CacheListener[]
    {listener1, listener2, listener3})
  .setScope(Scope.DISTRIBUTED_NO_ACK)
  .create(name);
```

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 regions that use the WBCL
  <region name="data">
    <region-attributes async-event-queue-ids="sampleQueue">
    </region-attributes>
  </region>
</cache>
```

Configuring Peer-to-Peer Event Messaging

This topic explains how to configure peer-to-peer event messaging in VMware Tanzu GemFire.

You can receive events from cluster 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.

> You can also configure Regions using the gfsh command-line interface. See Region Commands.

Configuring Client/Server Event Messaging

This topic explains how to configure client/server event messaging in VMware Tanzu GemFire.

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>.
2. Program the client to register interest in the entries you need.

   > 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 region'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.registerInterestForKeys(list, InterestResultPolicy.NONE, false);

// Register interest in all keys and download all available keys now
Region region3 = . . . ;
region3.registerInterestForAllKeys(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'.

3. For highly available event messaging, configure server redundancy. See Configuring Highly Available Servers.
4. To have events enqueued for your clients during client downtime, configure durable client/server messaging.
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. 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

This topic explains how to configure highly-available servers in VMware Tanzu GemFire.

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 deactivated.
> 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

This topic explains highly-available client/server event messaging in VMware Tanzu GemFire.

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 queues 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 gfsh alterruntime command

Set the interval for queue synchronization messages as follows:

- gfsh:

```
gfsh>alter runtime --message-sync-interval=2
```

- XML:

```
<!-- Set synchronization interval to 2 seconds -->
<cache ... message-sync-interval="2" />
```

- Java:

```
cache = CacheFactory.create();
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

This topic explains how to implement durable client/server event messaging in VMware Tanzu GemFire.

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 event subscriptions as durable. Events for durable queries and subscriptions are saved in a 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 event subscriptions.

These are the high-level tasks described in this topic:

1. Configure your client as durable
2. Decide which subscriptions should be durable and configure accordingly
3. Program your client to manage durable messaging for disconnect, reconnect, and event handling

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);
CacheFactory cf = new CacheFactory(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 cluster.

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 Tanzu 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.

> 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.

1. Disconnect with a request to keep your queues active by using Pool.close or ClientCache.close with the boolean keepalive parameter.

   ```
   clientCache.close(true);
   ```

   To be retained during client down time, durable continuous queries (CQs) must be executing at the time of disconnect.

2. Program your durable client's reconnection to:

    1. If desired, detect whether the previously registered subscription queue is available upon durable client reconnection and the count of pending events in the queue. Based on the results, you can then decide whether to receive the remaining events or close the cache if the number is too large.

       For example, for a client with only the default pool created:

       ```
       int pendingEvents = cache.getDefaultPool().getPendingEventCount();

       if (pendingEvents == -2) { // client connected for the first time ... // continue
       }
       else if (pendingEvents == -1) { // client reconnected but after the timeout period
       ... // handle possible data loss
       }
       else { // pendingEvents >= 0
       ... // decide to invoke readyForEvents() or ClientCache::close(false)/pool.destroy()
       }
       ```

       For a client with multiple pools:

       ```
       int pendingEvents = 0;

       int pendingEvents1 = PoolManager.find("pool1").getPendingEventCount();

       pendingEvents += (pendingEvents1 > 0) ? pendingEvents1 : 0;

       int pendingEvents2 = PoolManager.find("pool2").getPendingEventCount();

       pendingEvents += (pendingEvents2 > 0) ? pendingEvents2 : 0;

       // process individual pool counts separately.
       ```

       The getPendingEventCount API can return the following possible values: - A value representing a count of events pending at the server. Note that this count is an approximate value based on the time the durable client pool connected or reconnected to the server. Any number of invocations will return the same value. - A zero value if there are no events pending at server for this client pool - A negative value indicates that no queue is available at the server for the client pool. - -1 indicates that the client pool has reconnected to the server after its durable-client-timeout period has elapsed. The pool's subscription queue has been removed possibly causing data loss. - A value of -2 indicates that this client pool has connected to server for the first time.

    2. Connect, initialize the client cache, regions, and any cache listeners, and create and execute any durable continuous queries.

    3. Run all interest registration calls.

       > 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, deactivate concurrency checking for the region in the client cache. See Consistency for Region Updates.

4. 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 that are not defined in the cache.xml . . .
// Here, run all register interest calls before doing anything else
clientCache.readyForEvents();
```

3. When you program your durable client CacheListener:
    1. 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.
    2. 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.

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 and listeners 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 information until the client reconnects or times out. The non-durable interest lists are discarded. The servers continue to queue up incoming messages for entries on the durable interest 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, 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 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.
- Client cache operations by the application.
- Callbacks triggered by replaying old events from the queue

Tanzu GemFire handles the conflicts between the application and interest registrations so they do not create cache update conflicts. But you must program your event handlers so they do not 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

This topic explains how to tune client/server event messaging in VMware Tanzu GemFire.

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 cluster. 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.

Server-to-client message tracking example as described.

The client's message tracking list holds the highest sequence ID of any message received for each originating thread. The list can become excessively 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

This topic explains how to conflate the server subscription queue in VMware Tanzu GemFire.

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 deactivated by default.

Conflation is particularly useful when a single entry is updated often and the intermediate updates do not 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.

> 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 cluster.

Limit the Server's Subscription Queue Memory Use

This topic explains how to limit server subscription queue memory use in VMware Tanzu GemFire.

Options for limiting the amount of server memory the subscription queues consume:

- Optional: Conflate the subscription queue messages.
- Optional: Increase the frequency of queue synchronization by decreasing the pool's configuration parameter subscription-ack-interval. This applies only to configurations where server redundancy is used for high availability. Example:

```
<!-- Set subscription ack interval to 3 seconds -->
<cache>
  <pool ... subscription-enabled="true"
          subscription-ack-interval="3000">
  ...
  </pool>
...
```

  The client periodically sends an acknowledgment (ack) message to the server. Each message acknowledges the receipt of many events by the client. Since the server must retain every outbound event in the queue until its receipt is acknowledged, shortening the acknowledgment delay can reduce the average queue size, reducing the amount of server memory used for queueing.

- 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();
```

> 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

This topic explains how to tune client subscription message tracking timeout in VMware Tanzu GemFire.

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 Multi-Site (WAN) Event Queues

This topic explains how to configure multi-site (WAN) event queues in VMware Tanzu GemFire.

In a multi-site (WAN) installation, Tanzu 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.

> You can configure multiple regions to share the same AsyncEventQueue if they use the same AsyncEventListener. The maximum combined number of AsyncEventQueues and GatewaySenders per cluster is 128.

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 or Implementing an AsyncEventListener for Write-Behind Cache Event Handling.

- 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.

- Configuring Dispatcher Threads and Order Policy for Event Distribution

  By default, Tanzu GemFire uses multiple dispatcher threads to process region events simultaneously in a gateway sender queue for distribution between sites, or in an asynchronous event queue for distributing events for write-behind caching. With serial queues, you can also configure the ordering policy for dispatching those events.

- 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.

Persisting an Event Queue

This topic explains persist event queues in VMware Tanzu GemFire.

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. Tanzu GemFire persists an event queue if you set the enable-persistence 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.

You must configure the event queue to use persistence if you are using persistent regions. The use of non-persistent event queues with persistent regions is not supported.

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.

> If you configure a parallel queue and/or you configure multiple dispatcher threads for a 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 the gateway sender queue uses “diskStoreA” for persistence and overflow, and the queue has a maximum queue memory of 100 MB:

- XML example:

```
<cache>
  <gateway-sender id="persistedsender1" parallel="false"
    remote-distributed-system-id="1"
    enable-persistence="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();
```

- gfsh:

```
gfsh>create gateway-sender --id="persistedsender1 --parallel=false
--remote-distributed-system-id=1 --enable-persistence=true --disk-store-name=diskStoreA
--maximum-queue-memory=100
```

If you were to configure 10 dispatcher threads for the serial gateway sender, then the total maximum memory for the gateway sender queue would be 1000MB on each Tanzu GemFire member that hosted the sender, because Tanzu GemFire creates a separate copy of the queue per thread..

> You can configure multiple regions to share the same AsyncEventQueue if they use the same AsyncEventListener. The maximum combined number of AsyncEventQueues and GatewaySenders per cluster is 128.

Configuring Persistence for Async Event Queues

The following example shows a 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>gfeadmin</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);
```

- gfsh:

```
gfsh>create async-event-queue --id="persistentAsyncQueue" --persistent=true
--disk-store="diskStoreA" --parallel=true --listener=MyAsyncEventListener
--listener-param=url#jdbc:db2:SAMPLE --listener-param=username#gfeadmin --listener-param=password#admin1
```

Configuring Dispatcher Threads and Order Policy for Event Distribution

This topic explains how to configure dispatcher threads and order policy for event distribution in VMware Tanzu GemFire.

By default, Tanzu GemFire uses multiple dispatcher threads to process region events simultaneously in a gateway sender queue for distribution between sites, or in an asynchronous event queue for distributing events for write-behind caching. With serial queues, you can also configure the ordering policy for dispatching those events.

By default, a gateway sender queue or asynchronous event queue uses 5 dispatcher threads per queue. This provides support for applications that have the ability to process queued events concurrently for distribution to another Tanzu GemFire site or listener. If your application does not require concurrent distribution, or if you do not have enough resources to support the requirements of multiple dispatcher threads, then you can configure a single dispatcher thread to process a queue.

- Using Multiple Dispatcher Threads to Process a Queue
- Performance and Memory Considerations
- Configuring the Ordering Policy for Serial Queues

- Examples—Configuring Dispatcher Threads and Ordering Policy for a Serial Gateway Sender Queue

Using Multiple Dispatcher Threads to Process a Queue

When multiple dispatcher threads are configured for a parallel queue, Tanzu GemFire simply uses multiple threads to process the contents of each individual queue. The total number of queues that are created is still determined by the number of Tanzu GemFire members that host the region.

When multiple dispatcher threads are configured for a serial queue, Tanzu GemFire creates an additional copy of the queue for each thread on each member that hosts the queue. To obtain the maximum throughput, 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 a serial gateway sender or an asynchronous event queue uses multiple dispatcher threads, consider the following:

- Queue attributes are repeated for each copy of the 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 copy of the serial queue. If you configure 10 dispatcher threads and the maximum queue memory is set to 100MB, then the total maximum queue memory for the queue is 1000MB on each member that hosts the queue.

Configuring the Ordering Policy for Serial Queues

When using multiple dispatcher-threads (greater than 1) with a serial event 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. Tanzu 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. Tanzu 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 partitioning. With partition ordering, all entries that share the same “partitioning key” (RoutingObject) are placed into the same dispatcher thread queue.

You cannot configure the order-policy for a parallel event queue, because parallel queues cannot preserve event ordering for regions. Only the ordering of events for a given partition (or in a given queue of a distributed region) can be preserved.

Examples—Configuring Dispatcher Threads and Ordering Policy for a Serial Gateway Sender Queue

To increase the number of dispatcher threads and set the 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"
    enable-persistence="true"
    disk-store-name="gateway-disk-store"
    maximum-queue-memory="200"
    dispatcher-threads=7 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(7);
gateway.setOrderPolicy(OrderPolicy.KEY);
GatewaySender sender = gateway.create("NY", "1");
sender.start();
```

- gfsh:

```
gfsh>create gateway-sender -d="NY"
  --parallel=false
  --remote-distributed-system-id="1"
  --enable-persistence=true
  --disk-store-name="gateway-disk-store"
  --maximum-queue-memory=200
  --dispatcher-threads=7
  --order-policy="key"
```

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=7 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>
```

- 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(7);
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.

- gfsh:

```
gfsh>create async-event-queue --id="sampleQueue" --persistent=true
--disk-store="async-disk-store" --parallel=false
--dispatcher-threads=7 order-policy="key"
--listener=myAsycEventListener
--listener-param=url#jdbc:db2:SAMPLE
--listener-param=username#gfeadmin
--listener-param=password#admin1
```

Conflating Events in a Queue

This topic explains how to conflate events in a queue in VMware Tanzu GemFire.

Conflating a queue improves distribution performance. When conflation is enabled, only the latest queued value is sent for a particular key.

> 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 is removed and the new update is added to the end of the queue, as shown here for key A.

Existing update message for Key A in queue removed and new update for Key A added to end of queue

> This method of conflation is different from the one used for server-to-client subscription queue conflation and peer-to-peer distribution within a cluster.

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"
   enable-persistence="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.

- gfsh:

```
gfsh>create gateway-sender --id="NY" --parallel=true
   --remote-distributed-system-id="1"
   --enable-persistence=true
   --disk-store-name="gateway-disk-store"
   --enable-batch-conflation=true
```

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.

- gfsh:

```
gfsh>create async-event-queue --id="sampleQueue" --persistent=true
--disk-store="async-disk-store" --parallel="false"
--listener=myAsyncEventListener
--listener-param=url#jdbc:db2:SAMPLE
--listener-param=username#gfeadmin
--listener-param=password#admin1
```

Delta Propagation

This topic links to documentation about delta propagation in VMware Tanzu GemFire.

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.

- 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 with higher redundancy levels generally benefit more from delta propagation. However, in some application scenarios, delta propagation does not show any significant benefits. On occasion it results in performance degradation.

- Delta Propagation Properties

  This topic describes the properties that can be used to configure delta propagation.

- Implementing Delta Propagation

  By default, delta propagation is enabled in your cluster. When enabled, delta propagation is used for objects that implement org.apache.geode.Delta. You program the methods to store and extract delta information for your entries and to apply received delta information.

- Errors In Delta Propagation

  This topic lists the errors that can occur when using delta propagation.

- Delta Propagation Example

  This topic provides an example of delta propagation.

How Delta Propagation Works

This topic explains how delta propagation works in VMware Tanzu GemFire.

Delta propagation reduces the amount of data you send over the network. You do this by sending only 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.

Tanzu GemFire propagates object deltas using methods that you program. The methods are in the org.apache.geode.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 is not 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.
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 org.apache.geode.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.

> Only the object itself being placed in the cache can implement the Delta interface and propagate changes. Any sub-objects of the cache object do not propagate their changes.

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.

Tanzu GemFire also does not propagate deltas for:

- Transactional commit
- The putAll operation
- JVMs running Tanzu 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. Tanzu 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. Tanzu GemFire clients can always send deltas to the servers, and servers can usually sent 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). If a domain object implements the Delta interface, delta updates to it will be sent between sites. The ideal use case for this feature is to have each site start with the same entries so that when deltas are sent from one site to the other, they can be applied directly.

  If a delta is being processed by a GatewayReceiver in a remote site and either of these cases are true, then the full value will be sent by the originating GatewaySender:
    - The remote site does not have the value to which the delta must be applied
    - The domain object's fromDelta method throws an InvalidDeltaException

In both of those cases, an InvalidDeltaException for that key will be sent back to the originating site, and an event containing the full value for that key will be enqueued at the back of the GatewaySender's queue. Any deltas received after that full value event is processed by the remote site will be applied to that value.

> If the domain object's fromDelta method throws any other kind of exception, the full value will not be sent.

The full value will always be sent in these cases even if the delta is available:

- If either the GatewaySender or GatewayReceiver is running in a member whose version of GemFire earlier than v10.0
- If the member running either the GatewaySender or GatewayReceiver configures delta-propagation=false
- If the GatewaySender has conflation enabled
- If the partitioned region processing the deltas is attached to a GatwaySender configured with parallel=false
- Any update event that contains delta bytes but is treated like a create event (e.g. if the entry was destroyed while the update was occurring)
- Any event that occurs before the GatewaySender has a connection to the GatewayReceiver (since the connection is required to establish the GatewayReceiver's GemFire version)
- If the GatewaySender starts first, it will not have a connection to the GatewayReceiver. Any events created in that state will have the full value because the remote site version is unknown at that time. Similarly, if the GatewaySender starts first, followed by the GatewayReceiver, and the GatewaySender is then paused before any operations occur, no connection is established and all events will be sent as full values (as the GemFire version of the GatewayReceiver's member is unknown at the time the operations are put into the Gateway queue).

When to Avoid Delta Propagation

This topic explains when to avoid delta propagation when using VMware Tanzu GemFire.

Generally, the larger your objects and the smaller the deltas, the greater the benefits of using delta propagation. Partitioned regions with higher redundancy levels generally benefit more from delta propagation. However, in some application scenarios, delta propagation does not show any significant benefits. On occasion it results in performance degradation.

By default, delta propagation is enabled in your cluster.

These are the main factors that can reduce 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 deserialized form. This aspect of delta propagation only negatively impacts your system if your objects are not already being deserialized for other reasons, such as for indexing and querying or for listener operations. Once stored in deserialized 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 deserializing 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

This topic describes the properties that can be used to configure delta propagation in VMware Tanzu GemFire.

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 activates or deactivates delta propagation. When false, full entry values are sent for every update. The default setting is true, which enables delta propagation.

Deactivate 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.
- If the Region attribute off-heap is true, cloning is enabled, regardless of what this attribute is set to.

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 Tanzu 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.
- Tanzu 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");
```

- gfsh:

```
gfsh>create region --name="region_with_cloning" --type=REPLICATE
--enable-cloning=true
```

Implementing Delta Propagation

This topic explains how to implement delta propagation in VMware Tanzu GemFire.

By default, delta propagation is enabled in your cluster. When enabled, delta propagation is used for objects that implement org.apache.geode.Delta. You program the methods to store and extract delta information for your entries and to apply received delta information.

Use the following procedure to implement delta propagation in your cluster.

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.
2. For each region where you are using delta propagation, choose whether to enable cloning using the delta propagation property cloning-enabled. Cloning is deactivated by default. See Delta Propagation Properties.
3. If you do not enable cloning, review all associated listener code for dependencies on EntryEvent.getOldValue. Without cloning, Tanzu 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 org.apache.geode.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:
    1. 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.
    2. 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.
    3. 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.
    4. 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 Tanzu 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.
    5. 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.
    6. Write hasDelta to report on whether a delta is available.
    7. 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.
    8. Write fromDelta to decode the byte stream that toDelta creates and update the object.
    9. 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

This topic lists the errors that can occur when using delta propagation in VMware Tanzu GemFire.

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

This topic provides an example of delta propagation in VMware Tanzu GemFire.

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, Tanzu GemFire calls hasDelta, which returns true, so Tanzu GemFire calls toDelta and forwards the extracted delta to the server. If hasDelta returned false, Tanzu GemFire would distribute the full entry value.
2. In Server1, Tanzu 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, Tanzu 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 org.apache.geode.Delta;
import org.apache.geode.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 intVal;
    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.intVal = 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 about what has changed to the
    // data stream, so fromDelta knows what it is getting
        out.writeBoolean(intFldChd);
        if (intFldChd) {
    // Write just the changes into the data stream
            out.writeInt(this.intVal);
    // Once the delta information is written, reset the delta status field
            this.intFldChd = false;
            System.out.println(" Extracted delta from field 'intVal' = "
                                    + this.intVal);
        }
        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 + "} ]";
    }
}
```

Querying

This topic provides links to documentation about querying in VMware Tanzu GemFire.

Tanzu GemFire provides a SQL-like querying language, Object Query Language (OQL), that allows you to access data stored in Tanzu GemFire regions.

Because Tanzu GemFire regions are key-value stores where values can range from simple byte arrays to complex nested objects, Tanzu GemFire uses a query syntax based on OQL to query region data. OQL is very similar to SQL, but OQL allows you to query complex objects, object attributes, and methods.

- Querying FAQ and Examples

  This topic answers some frequently asked questions on querying functionality. It provides examples to help you get started with Tanzu GemFire querying.

- Querying with OQL

  This section provides a high-level introduction to Tanzu GemFire querying such as building a query string and describes query language features.

- Writing and Executing a Query in OQL

  This section explains how to create a query object and use it to perform query-related operations.

- Advanced Querying

  This section includes advanced querying topics such as using query indexes, using query bind parameters, querying partitioned regions and query debugging.

- Working with Indexes

  The Tanzu GemFire query engine supports indexing. An index can provide significant performance gains for query execution.

Querying FAQ and Examples

This topic answers frequently asked questions about querying functionality in VMware Tanzu GemFire and provides examples to help you get started with Tanzu GemFire querying.

For additional information about Tanzu GemFire querying, see Querying.

- How do I write and execute a query against a Tanzu GemFire region?
- Can I see query string examples, listed by query type?
- Which APIs should I use to write my queries?
- How do I invoke an object's method in a query?
- Can I invoke a static method on an object in a query?
- How do I write a reusable query?
- When should I create indexes to use in my queries?
- How do I create an index?
- Can I query a partitioned region? Can I perform a join query on a partitioned region?
- How can I improve the performance of a partitioned region query?
- Which query language elements are supported in Tanzu GemFire?
- How do I debug queries?
- Can I use implicit attributes or methods in my query?
- How do I perform a case-insensitive search on a field in OQL?

How do I write and execute a query against a Tanzu GemFire region?

To write and execute a query in Tanzu GemFire, you can use any of the following mechanisms. Sample query code follows.

- Tanzu GemFire querying APIs
- gfsh command-line interface; in particular the query command
- REST API query endpoints

Sample Tanzu 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. */
```

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;
}
```

Basic WHERE Clause Examples

In the following examples, the status field is type String and the ID field is type int. See Supported Literals for a complete list of literals supported in Tanzu GemFire querying.

1. Select all active portfolios.

   ```
   SELECT * FROM /exampleRegion WHERE status = 'active'
   ```

2. Select all portfolios whose status begins with ‘activ’.

   ```
   SELECT * FROM /exampleRegion p WHERE p.status LIKE 'activ%'
   ```

3. 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 = 'active'
```

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 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 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
```

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 MIN and MAX

See MIN and MAX for more information.

```
SELECT MIN(ID)
FROM /exampleRegion
WHERE ID > 0
```

```
SELECT MAX(ID)
FROM /exampleRegion
WHERE ID > 0 AND status LIKE 'act%'
```

```
SELECT MIN(pos.mktValue)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID = 10
```

```
SELECT MAX(p.ID)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 OR p.status = 'active' OR pos.secId = 'IBM'
```

Using AVG

See AVG for more information.

```
SELECT AVG(ID)
FROM /exampleRegion
WHERE ID > 0
```

```
SELECT AVG(ID)
FROM /exampleRegion
WHERE ID > 0 AND status LIKE 'act%'
```

```
SELECT AVG(pos.mktValue)
FROM /exampleRegion p, p.positions.values pos
WHERE p.isActive()
```

```
SELECT AVG(DISTINCT p.ID)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 OR p.status = 'active' OR pos.secId = 'IBM'
```

Using COUNT

See COUNT 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(DISTINCT p.status)
FROM /exampleRegion p
WHERE p.ID > 0
```

```
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 = 'IBM'
```

Using SUM

See SUM for more information.

```
SELECT SUM(ID)
FROM /exampleRegion
WHERE ID > 0
```

```
SELECT SUM(ID)
FROM /exampleRegion
WHERE ID > 0 AND status LIKE 'act%'
```

```
SELECT SUM(pos.mktValue)
FROM /exampleRegion p, p.positions.values pos
WHERE p.status = 'active'
```

```
SELECT SUM(DISTINCT p.ID)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 OR p.status = 'active' OR pos.secId = 'IBM'
```

Using GROUP BY

See GROUP BY for more information.

```
SELECT p.status, MAX(p.ID)
FROM /exampleRegion p
WHERE p.ID > 0
GROUP BY p.status
```

```
SELECT p.ID, MIN(pos.qty) AS lessQty
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 AND p.status = 'active'
GROUP BY p.ID
ORDER BY lessQty ASC
```

```
SELECT p.ID, MAX(pos.mktValue) AS maxValue
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 AND p.status = 'active'
GROUP BY p.ID
ORDER BY maxValue DESC
```

```
SELECT p.status, AVG(p.ID)
FROM /exampleRegion p
WHERE p.ID > 0
GROUP BY p.status
```

```
SELECT p.ID, pos.secId, AVG(pos.mktValue)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 OR p.status = 'active'
GROUP BY p.ID, pos.secId
```

```
SELECT p.status, AVG(p.ID) as sm
FROM /exampleRegion p
WHERE p.ID > 0
GROUP BY p.status
ORDER BY sm DESC
```

```
SELECT p.status, COUNT(*)
FROM /exampleRegion p
WHERE p.ID > 0
GROUP BY p.status
```

```
SELECT p.ID, COUNT(pos) AS positionsAmount
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 OR p.status = 'active'
GROUP BY p.ID
ORDER BY positionsAmount
```

```
SELECT p.status, SUM(p.ID)
FROM /exampleRegion p
WHERE p.ID > 0
GROUP BY p.status
```

```
SELECT p.ID, pos.secId, SUM(pos.mktValue)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 OR p.status = 'active'
GROUP BY p.ID, pos.secId
```

```
SELECT p.status, SUM(p.ID) as sm
FROM /exampleRegion p
WHERE p.ID > 0
GROUP BY p.status
ORDER BY sm DESC
```

```
SELECT p.ID, SUM(pos.mktValue) AS marketValue
FROM /exampleRegion p, p.positions.values pos
WHERE p.isActive()
GROUP BY p.ID
ORDER BY marketValue DESC
```

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%
```

Using Region Entry Keys and Values

```
SELECT * FROM /exampleRegion.keys k WHERE k.ID = 1
```

```
SELECT entry.value FROM /exampleRegion.entries entry WHERE entry.key = '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 * 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

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]
```

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 /exampleRegoin1 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.

```
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', '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 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 namescopes (packages), there needs to be a means of referring to different classes of the same name. The IMPORT statement is used to establish a namescope for a class in a query.

```
IMPORT package.Position;
SELECT DISTINCT * FROM /exampleRegion, positions.values positions TYPE Position WHERE positions.mktValue >= 25.00
```

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 org.apache.geode.cache.Cache.getQueryService.

If you are writing a Java client to server query, use org.apache.geode.cache.client.Pool.getQueryService.

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:

```
// specify the  query string
 String queryString = "SELECT DISTINCT * FROM /exampleRegion p WHERE p.status = $1";

QueryService queryService = cache.getQueryService();
Query query = queryService.newQuery(queryString);

// set a query bind parameter
Object[] params = new Object[1];
params[0] = "active";

// Execute the query locally. It returns the results set.
SelectResults results = (SelectResults) query.execute(params);

// use the results of the query; this example only looks at the size
 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 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 index?

An 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 about using this API, see the 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 about indexes, see Working with Indexes.

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 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 colocated partitioned regions by executing a function on a local data set.

For a full list of restrictions, see Partitioned Region Query Restrictions.

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 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.

Which query language elements are supported in Tanzu GemFire?

AND	IN	ORDER BY
AS	IS_DEFINED	SELECT
COUNT	IS_UNDEFINED	SET
DISTINCT	LIMIT	<TRACE>
ELEMENT	LIKE	TRUE
FROM	NOT	TO_DATE
<HINT>	NVL	TYPE
IMPORT	OR	WHERE

For more information and examples on using each supported keyword, see Supported Keywords.

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, Tanzu 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");
```

Can I use implicit attributes or methods in my query?

If an implicit attribute or method name can only be associated with one untyped iterator, the Tanzu GemFire query processor will assume that it is associated with that iterator. However, if more than one untyped iterator is in scope, then the query will fail with a TypeMismatchException. The following query fails because the query processor does not fully type expressions:

```
select distinct value.secId from /pos , getPositions(23)
```

The following query, however, succeeds because the iterator is either explicitly named with a variable or it is typed:

```
select distinct e.value.secId from /pos , getPositions(23) e
```

Can I instruct the query engine to use specific indexes with my queries?

Using HINT *indexname* allows you to instruct the query engine to prefer and filter results from the specified indexes. If you provide multiple index names, the query engine will use all available indexes but prefer the specified indexes.

```
<HINT 'IDIndex'> SELECT * FROM /Portfolios p WHERE p.ID > 10 AND p.owner = 'XYZ'
```

```
<HINT 'IDIndex', 'OwnerIndex'> SELECT * FROM /Portfolios p WHERE p.ID > 10 AND p.owner = 'XYZ' AND p.value < 100
```

How do I perform a case-insensitive search on a field in OQL?

You can use the Java String class methods toUpperCase and toLowerCase to transform fields where you want to perform a case-insensitive search. For example:

```
SELECT entry.value FROM /exampleRegion.entries entry WHERE entry.value.toUpperCase LIKE '%BAR%'
```

or

```
SELECT * FROM /exampleRegion WHERE foo.toLowerCase LIKE '%bar%'
```

Querying with OQL

This topic provides a high-level introduction to VMware Tanzu GemFire querying, query language features, and describes building a query string.

Tanzu GemFire provides a SQL-like querying language that allows you to access data stored in Tanzu GemFire regions. Because Tanzu GemFire regions are key-value stores where values can range from simple byte arrays to complex nested objects, Tanzu GemFire uses a query syntax based on Object Query Language (OQL) to query region data. OQL and SQL have many syntactical similarities, but also have significant differences. For example, OQL does not offer all of the capabilities of SQL like aggregates, but OQL does allow you to execute queries on complex object graphs, query object attributes, and invoke object methods.

The syntax of a typical Tanzu GemFire OQL query is as follows:

```
[IMPORT package]
SELECT [DISTINCT] projectionList
FROM collection1, [collection2, ...]
[WHERE clause]
[ORDER BY order_criteria [desc]]
```

A simple Tanzu GemFire OQL query resembles the following:

```
SELECT DISTINCT * FROM /exampleRegion WHERE status = 'active'
```

An important characteristic of Tanzu GemFire querying to note is that, by default, Tanzu 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 more information about Tanzu GemFire querying, see the Tanzu GemFire Querying FAQ and Examples.

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 OQL

This topic explains how to write and execute a query using the VMware Tanzu GemFire QueryService.

The Tanzu 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 org.apache.geode.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 org.apache.geode.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

> You can also perform queries using the gfsh query command. See query.

Next Steps

- Building a Query String: To build a query string, combine supported keywords, expressions, and operators to create an expression that returns the information that you require.
- OQL Syntax and Semantics: This section describes the querying language features.
- Query Language Restrictions and Unsupported Features: This section describes some limitations to be aware of when composing queries in Object Query Language (OQL).

Building a Query String

This topic explains how to build a query string in VMware Tanzu GemFire.

A query string is a fully-formed Object Query Language (OQL) statement that can be passed to a query engine and executed against a data set. The OQL statement must be a `SELECT` expression that references at least one region.

To build a query string, combine supported keywords, expressions, and operators to create an expression that returns the information that 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.
- Path Expressions: For example, in the query `SELECT * FROM /exampleRegion,/exampleRegion` is a path expression. See FROM Clause.
- 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.
- 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.
- Operators: Comparison operators (=,<,>,<>), unary operators (NOT), logical operators (AND, OR), and others. See Operators for a complete list.
- Literals: Boolean, date, time, and others. See Supported Literals 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 for more details.
- Preset Query Functions: For example, `ELEMENT(expr)` and `IS_DEFINED(expr)`. See SELECT Statement for other available functions.
- SELECT Statements: For example, in the example queries above `SELECT *` or `SELECT DISTINCT *`. See SELECT Statement for other available functions.
- Comments: OQL permits extra characters to accompany the query string without changing the string's definition. Form a multi-line comment by enclosing the comment body within `/*` and `*/` delimiters. OQL does not permit nested comments. A single line comment body consists of all of the characters to the right of `--` (two hyphens) on a single line.

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 Tanzu GemFire queries.

- IMPORT Statement
- FROM Clause
- WHERE Clause

- SELECT Statement
- OQL Aggregate Functions

IMPORT Statement

This topic explains how to use the IMPORT statement in queries in VMware Tanzu GemFire.

It may be necessary for an OQL query to refer to the class of an object. In cases where the same class name resides in two different namescopes (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

This topic explains how to use the FROM clause in queries in VMware Tanzu GemFire.

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:

Query a region for all distinct values. Return a collection of unique entry values from the region:

```
SELECT DISTINCT * FROM /exampleRegion
```

Query the top level region data using entrySet. Return the keys and positions of Region.Entry objects whose mktValue attribute is greater than 25.00:

```
SELECT key, positions FROM /exampleRegion.entrySet, value.positions.values positions WHERE positions.mktValue >= 25.00
```

Query the region for its entry values. Return a set of unique values from Region.Entry objects that have the key equal to 1:

```
SELECT DISTINCT entry.value FROM /exampleRegion.entries entry WHERE entry.key = '1'
```

Query the region for its entry values. Return the set of all entry values in which the ID field is greater than 1000:

```
SELECT * FROM /exampleRegion.entries entry WHERE entry.value.ID > 1000
```

Query entry keys in the region. Return a set of entry keys in the region that have the key equal to '1':

```
SELECT * FROM /exampleRegion.keySet key WHERE key = '1'
```

Query values in the region. Return a collection of entry values in the region that have the status attribute value of 'active':

```
SELECT * FROM /exampleRegion.values portfolio WHERE portfolio.status = 'active'
```

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

This topic explains how to use the WHERE clause in queries in VMware Tanzu GemFire.

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 of the entry 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 p WHERE p.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.

First example query:

```
SELECT * FROM /CustomObjects c
WHERE c.ID > 1 AND c.ID < 3
AND c.otherValue > 0 AND c.otherValue < 3
```

This query returns two objects, two of either CustomObjectA or CustomObjectB.

Second example query:

```
SELECT * FROM /CustomObjects c
WHERE c.ID > 1 AND c.ID < 3
AND c.otherValue > 1 AND c.otherValue < 3
```

This query returns either zero results or two results of CustomObjectB, depending on which entry is evaluated last.

To avoid unpredictable querying behavior, implement detailed versions of the equals and hashCode methods.

If you are comparing a non-primitive field of the object in the WHERE clause, use the equals method instead of the = operator. For example instead of nonPrimitiveObj = objToBeCompared use nonPrimitiveObj.equals(objToBeCompared).

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 gfsh, execute the following command before starting up your data members:

```
gfsh>configure pdx --read-serialized=true
```

See configure pdx for more information.

In cache.xml, set the following:

```
// 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 about indexes, see Working with Indexes.

Examples:

Query two regions. Return the ID and status for portfolios that have the same status.

```
SELECT portfolio1.ID, portfolio2.status FROM /exampleRegion portfolio1, /exampleRegion2 portfolio2 WHERE portfolio1.status
= portfolio2.status
```

Query two regions, iterating over all positions within each portfolio. Return all 4-tuples consisting of the value from each of the two regions and the value portion of the positions map from both regions in which the secId field of positions match.

```
SELECT * FROM /exampleRegion portfolio1, portfolio1.positions.values positions1, /exampleRegion2 portfolio2, portfolio2.positi
ons.values positions2 WHERE positions1.secId = positions2.secId
```

Same query as the previous example, with the additional constraint that matches will have a ID of 1.

```
SELECT * FROM /exampleRegion portfolio1, portfolio1.positions.values positions1, /exampleRegion2 portfolio2, portfolio2.positi
ons.values positions2 WHERE portfolio1.ID = 1 AND positions1.secId = positions2.secId
```

LIKE

Tanzu GemFire offers limited support for the LIKE predicate. LIKE can be used to mean ‘equals to’. If you terminate the string with a wildcard (‘%’), 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.

> The ‘*’ wildcard is not supported in OQL LIKE predicates.

You can also use the LIKE predicate when an index is present.

Examples:

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%'
```

Case Insensitive Fields

You can use the Java String class methods toUpperCase and toLowerCase to transform fields where you want to perform a case-insensitive search. For example:

```
SELECT entry.value FROM /exampleRegion.entries entry WHERE entry.value.toUpperCase LIKE '%BAR%'
```

or

```
SELECT * FROM /exampleRegion WHERE foo.toLowerCase LIKE '%bar%'
```

Method Invocations

To use a method in a query, use the attribute name that maps to the public method you want to invoke, or directly use the public method name instead. It is important to note that when you use the attribute name instead of the method name, VMware Tanzu GemFire will search for public methods named as the attribute itself or public methods with the get prefix.

```
SELECT r.id FROM /exampleRegion r                          - maps to object.id() or object.getId()
SELECT q.getName() FROM /exampleRegion q                   - maps to object.getName()
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 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.

Methods calls with the SecurityManager enabled

When the SecurityManager is enabled, by default Tanzu GemFire throws a NotAuthorizedException when any method that does not belong to the to the list of default allowed methods, given in RestrictedMethodAuthorizer, is invoked.

In order to further customize this authorization check, see Changing the Method Authorizer.

In the past you could use the system property gemfire.QueryService.allowUntrustedMethodInvocation to deactivate the check altogether, but this approach is deprecated and will be removed in future releases. You must configure the UnrestrictedMethodAuthorizer instead.

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 enum type using the toString method:

```
// eStatus is an enum with values 'active' and 'inactive'
select * from /exampleRegion p where p.eStatus.toString() = 'active'
```

Query enum type using a bind parameter. The value of the desired Enum field (Day.Wednesday) is passed as an execution parameter:

```
select distinct * from /QueryRegion0 where aDay = $1
```

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 contains only '20' and '21', then the query evaluates 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 Tanzu 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].

Examples where p.value and p.value1 are both NaN:

Query	Evaluates to	Appears in the result set
SELECT * FROM /positions p WHERE p.value = 0	false	no
SELECT * FROM /positions p WHERE p.value > 0	true	yes

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
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;
```

Arithmetic Operations

Arithmetic operators may be used in any expression.

For example, this query selects all people with a body mass index less than 25:

```
String query = "SELECT * FROM /people p WHERE p.height * p.height/p.weight < 25";
```

SELECT Statement

This topic explains how to use the SELECT statement in queries in VMware Tanzu GemFire.

The SELECT statement allows you to filter data from the collection of objects 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 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.utils.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

To limit the results set to unique rows, use the DISTINCT keyword. For example:

```
SELECT DISTINCT * FROM /exampleRegion
```

> 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
```

> If you are using ORDER BY queries, you must implement the equals and hashCode methods for the objects that you query.

Preset Query Functions

Tanzu 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. Inequality queries include undefined values in their query results. With the IS_DEFINED function, you can limit results to only those elements with defined values.	`IS_DEFINED(SELECT DISTINCT * FROM /exampleRegion p WHERE p.status = 'active')`
IS_UNDEFINED (expr)	Returns TRUE if the expression evaluates to UNDEFINED. With the exception of inequality queries, most queries do not include undefined values in their query results. The IS_UNDEFINED function allows undefined values to be included, so you can identify elements 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.	

Function	Description	Example
TO_DATE(date_str, format_str)	Returns a Java Date. 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.	

OQL Aggregate Functions

This topic provide descriptions and limitations about the aggregate functions for queries in VMware Tanzu GemFire, and explanations of how they work and how they can be used.

The following aggregate functions are supported:

- GROUP BY
- MIN, MAX, COUNT, and COUNT over a DISTINCT expression
- SUM and SUM over a DISTINCT expression
- AVG, and AVG over a DISTINCT expression

GROUP BY

You can collect data across multiple entries and group the results by one or more columns through the usage of the GROUP BY statement. Note the following facts about its usage:

- It can group results by one or more fields.
- It returns a single record / entry per group.
- It must always be placed before the ORDER BY clause, if any.
- It can be used in conjunction with other aggregate functions: MIN, MAX, COUNT, SUM, and AVG.
- It groups records using the selected fields if and only if the fields have identical data across entries.
- It is required, whenever an aggregate function is used within a query with other selected fields, to also use GROUP BY.
- If there are no other aggregate functions within the query, all fields included within a GROUP BY clause must also be part of the original projection list, and all fields included within the projection list must also be part of the GROUP BY clause.

The following are example GROUP BY queries.

The following GROUP BY query returns the maximum amount of sales per employee.

```
SELECT ID, MAX(e.sales)
FROM /employees e
GROUP BY ID
```

The following GROUP BY query returns the minimum, maximum, total count, average and summation of IDs grouped by status.

```
SELECT pf.status, MIN(pf.ID), MAX(pf.ID), COUNT(pf.ID), AVG(pf.ID), SUM(pf.ID)
FROM /portfolio pf
GROUP BY pf.status
```

MIN

The MIN keyword returns the minimum or smallest value from the selected expression. The expression itself must always evaluate to java.lang.Comparable. The MIN statement returns the actual type of the selected element as its result.

The following are example MIN queries that return region entries (the entries implement the java.lang.Comparable interface).

```
SELECT MIN(pf)
FROM /exampleRegion pf
```

```
SELECT MIN(pf)
FROM /exampleRegion
pf WHERE pf.ID > 0
```

```
SELECT MIN(pf)
FROM /exampleRegion pf
WHERE pf.ID > 10 LIMIT 50
```

```
SELECT MIN(pf)
FROM /exampleRegion pf
WHERE pf.ID > 0 AND pf.status LIKE 'act%'
```

The following MIN query returns the lowest entry ID that matches the query's selection criteria.

```
SELECT MIN(pf.ID)
FROM /exampleRegion pf, pf.positions.values pos
WHERE pf.ID > 0 AND pos.secId = 'IBM'
```

The following MIN query returns the lowest positive ID grouped by status.

```
SELECT pf.status, MIN(pf.ID)
FROM /exampleRegion pf
WHERE pf.ID > 0
GROUP BY pf.status
```

MAX

The MAX keyword returns the maximum or highest value from the selected expression. The expression itself must always evaluate to java.lang.Comparable. The MAX statement returns the type of the selected element as its result.

The following are example MAX queries that return region entries (the entries implement the java.lang.Comparable interface).

```
SELECT MAX(pf)
FROM /exampleRegion pf
```

```
SELECT MAX(pf)
FROM /exampleRegion
pf WHERE pf.ID > 0
```

```
SELECT MAX(pf)
FROM /exampleRegion pf
WHERE pf.ID > 10 LIMIT 50
```

```
SELECT MAX(pf)
FROM /exampleRegion pf
WHERE pf.ID > 0 AND pf.status LIKE 'act%'
```

The following MAX query returns the highest entry ID that matches the query's selection criteria.

```
SELECT MAX(pf.ID)
FROM /exampleRegion pf, pf.positions.values pos
WHERE pf.ID > 0 AND pos.secId = 'IBM'
```

The following MAX query returns the highest positive IDs grouped by status.

```
SELECT pf.status, MAX(pf.ID)
FROM /exampleRegion pf
WHERE pf.ID > 0
GROUP BY pf.status
```

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 a java.lang.Integer or java.lang.Long as the result (depending on how big the value is); you should take this into consideration when executing the query: if an overflow occurs while computing the COUNT function because the value is higher than Long.MAX_VALUE (2^{63} - 1), the result will be incorrect.

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 = 'IBM'
```

SUM

The SUM keyword returns the summation of all results that match the query selection conditions specified in the WHERE clause. Using SUM allows you to aggregate specific numeric values within a results set. For partitioned regions, each node's buckets compute a sum over that node and return the result to the coordinator node executing the query, which then aggregates the sums across all nodes.

The SUM function where the DISTINCT modifier is applied to the expression returns the summation over the set of unique (distinct) values. For partitioned regions, the distinct values in a node's buckets are returned to the coordinator node, which can then calculate the sum over the values that are unique across nodes, after eliminating duplicate values that come from separate nodes.

The actual expression used to calculate the aggregation should be an instance of java.lang.Number. The SUM statement always returns a java.lang.Number as the result and, depending on how big the value is and whether it has a decimal component or not, the returned type could be an instance of java.lang.Integer, java.lang.Long, java.lang.Float or java.lang.Double; you should take this into consideration when executing the query: if an overflow occurs while computing the SUM function because the value is higher than Double.MAX_VALUE ($(2 - 2^{-52}) * 2^{1023}$), the result will be incorrect.

The following are example SUM queries that return the summation of the entries ID.

```
SELECT SUM(ID)
FROM /exampleRegion
```

```
SELECT SUM(ID)
FROM /exampleRegion
WHERE ID > 0
```

```
SELECT SUM(ID)
FROM /exampleRegion
WHERE ID > 0 LIMIT 50
```

```
SELECT SUM(ID)
FROM /exampleRegion
WHERE ID > 0 AND status LIKE 'act%'
```

```
SELECT SUM(ID)
FROM /exampleRegion
WHERE ID IN SET(1,2,3,4,5)
```

The following SUM query returns the total summation of positive IDs grouped by status.

```
SELECT pf.status, SUM(pf.ID)
FROM /exampleRegion pf
WHERE pf.ID > 0
GROUP BY pf.status
```

The following SUM query uses the DISTINCT keyword and eliminates duplicates from the aggregation.

```
SELECT SUM(DISTINCT pf.ID)
FROM /exampleRegion pf, pf.positions.values pos
WHERE pf.ID > 0 OR pf.status = 'active' OR pos.secId = 'IBM'
```

The following SUM query returns the total aggregation of positive IDs grouped by status and sorted by the aggregation result in descending order.

```
SELECT pf.status, SUM(pf.ID) as sm
FROM /exampleRegion pf
WHERE pf.ID > 0
GROUP BY pf.status
ORDER BY sm DESC
```

AVG

The AVG keyword returns the arithmetic mean of the set formed by the selected expression. For partitioned regions, each node's buckets provide both a sum and the number of elements to the node executing the query (coordinator), such that a correct average may be computed.

The AVG keyword where the DISTINCT modifier is applied to the expression returns the arithmetic mean of the set of unique (distinct) values. For partitioned regions, the distinct values in a node's buckets are returned to the coordinator node, which can then calculate the average over the values that are unique across nodes, after eliminating duplicate values that come from separate nodes.

The actual expression used to calculate the aggregation should be an instance of java.lang.Number. The AVG statement always returns a java.lang.Number as the result and, depending on how big the value is and whether it has a decimal component or not, the returned type could be an instance of java.lang.Integer, java.lang.Long, java.lang.Float or java.lang.Double; you should take this into consideration when executing the query: if an overflow occurs while computing the AVG function because the value is higher than Double.MAX_VALUE ($(2 - 2^{-52}) * 2^{1023}$), or if an overflow occurs while computing the intermediate count because the amount of elements is higher than Long.MAX_VALUE ($2^{63} - 1$), the result will be incorrect.

The following are example AVG queries that calculate the average of the entries ID.

```
SELECT AVG(ID)
FROM /exampleRegion
```

```
SELECT AVG(ID)
FROM /exampleRegion
WHERE ID > 0
```

```
SELECT AVG(ID)
FROM /exampleRegion
WHERE ID > 0 LIMIT 50
```

```
SELECT AVG(ID)
FROM /exampleRegion
WHERE ID > 0 AND status LIKE 'act%'
```

```
SELECT AVG(ID)
FROM /exampleRegion
WHERE ID IN SET(1,2,3,4,5)
```

The following AVG query returns the average of positive IDs grouped by status.

```
SELECT pf.status, AVG(pf.ID)
FROM /exampleRegion pf
WHERE pf.ID > 0
GROUP BY pf.status
```

The following AVG query uses the DISTINCT keyword and eliminates duplicates from the aggregation.

```
SELECT AVG(DISTINCT pf.ID)
FROM /exampleRegion pf, pf.positions.values pos
WHERE pf.ID > 0 OR pf.status = 'active' OR pos.secId = 'IBM'
```

The following AVG query returns the average of positive IDs grouped by status and sorted by the calculation result in descending order.

```
SELECT pf.status, AVG(pf.ID) as sm
FROM /exampleRegion pf
WHERE pf.ID > 0
GROUP BY pf.status
ORDER BY sm DESC
```

OQL Syntax and Semantics

This topic links to documentation about VMware Tanzu GemFire query language features.

- Supported Character Sets
- Supported Keywords
- Case Sensitivity
- Comments in Query Strings
- Query Language Grammar
- Operators
- Reserved Words
- Supported Literals

Supported Character Sets

The VMware Tanzu GemFire query language supports the full ASCII and Unicode character sets.

Supported Keywords

VMware Tanzu GemFire supports the following 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 Operators
AS	Used to provide a label for a path expression so you can refer to the path by the label later.	See Aliases and Synonyms
AVG	Returns the average arithmetic mean of the set formed by the selected expression. The type of the expression must evaluate to a java.lang.Number.	See AVG
COUNT	Returns the quantity of values in the set formed by the selected expression.	See COUNT
DISTINCT	Restricts the select statement to unique results (eliminates duplicates).	See DISTINCT
ELEMENT	Query function. 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.	See Preset Query Functions
FROM	You can access any object or object attribute that is available in the current scope of the query.	See FROM Clause
GROUP BY	Allows you to summarize data, arranging or grouping results by identical fields.	See GROUP BY
<HINT>	Keyword that instructs the query engine to prefer certain indexes.	See Using Query Index Hints
IMPORT	Used to establish the name for a class in a query.	See IMPORT Statement
IN	The IN expression is a Boolean indicating whether one expression is present inside a collection of expressions of a compatible type.	See IN and SET
IS_DEFINED	Query function. Returns TRUE if the expression does not evaluate to UNDEFINED. Inequality queries include undefined values in their query results. With the IS_DEFINED function, you can limit results to only those elements with defined values.	See Preset Query Functions
IS_UNDEFINED	Query function. Returns TRUE if the expression evaluates to UNDEFINED. With the exception of inequality queries, most queries do not include undefined values in their query results. The IS_UNDEFINED function allows undefined values to be included, so you can identify elements with undefined values.	See Preset Query Functions
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 LIMIT
LIKE	LIKE can be used to mean ‘equals to’. If you terminate the string with a wildcard (‘%’), 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 can also use the LIKE predicate if an index is present.	See LIKE
MIN	Returns the minimum or smallest value from the selected expression. The type of the expression must evaluate to a java.lang.Comparable.	See MIN
MAX	Returns the maximum or largest value from the selected expression. The type of the expression must evaluate to a java.lang.Comparable.	See MAX

Query Language Keyword	Description	Example
NOT	The example returns the set of portfolios that have positions. Note that NOT cannot use an index.	See Operators
NVL	Returns expr2 if expr1 is null. The expressions can be query parameters (bind arguments), path expressions, or literals.	See Preset Query Functions
OR	If an expression uses both AND and OR operators, the AND expression has higher precedence than OR.	See Operators
ORDER BY	Allows you to order query results (either in ascending or descending order).	See ORDER BY
SELECT	Allows you to filter data from the collection of objects returned by a WHERE search operation.	See SELECT Statement
SET	Specifies a collection of values that can be compared to the returned values of query.	See IN and SET
SUM	Returns the summation over the set formed by the selected expression. The type of the expression must evaluate to a java.lang.Number.	See SUM
<TRACE>	Enables debugging on the following query string.	See Query Debugging
TO_DATE	Returns a Java Date. 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 Preset Query Functions
TYPE	Specifying object type in the FROM clause helps the query engine to process the query at optimal speed.	See Object Typing
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 WHERE Clause

Case Sensitivity

This topic explains case sensitivity when using queries in VMware Tanzu GemFire.

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.

In terms of query string and region entry matching, if you want to perform a case-insensitive search on a particular field, you can use the Java String class toUpperCase and toLowerCase methods in your query. For example:

```
SELECT entry.value FROM /exampleRegion.entries entry WHERE entry.value.toUpperCase LIKE '%BAR%'
```

or

```
SELECT * FROM /exampleRegion WHERE foo.toLowerCase LIKE '%bar%'
```

Comments in Query Strings

This topic explains how add comments to VMware Tanzu GemFire query strings.

Comment lines being 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

This topic describes the VMware Tanzu GemFire query language grammar.

Language Grammar

Notation used in the grammar:

nonterm *(all lower case)*

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 terminal symbols.

TERM *(all caps)*

A terminal symbol *(all caps)*. In the language these are case-insensitive keywords.

x y x followed by y

x | y x or y

(x | y) x or y

[x] x or empty

{ x } A possibly empty sequence of x.

// comment
descriptive text (in place of actual production rule)

Grammar rules:

```
queryProgram ::= [ traceCommand ] [ imports semicolon ] query [semicolon]
traceCommand ::= < TRACE >
imports ::= import { semicolon import }
import ::= IMPORT qualifiedName [ AS identifier ]
query ::= selectExpr | expr
selectExpr ::= [ hintCommand ] SELECT [ DISTINCT | ALL ] projectionAttributes fromClause [ whereClause ] [ groupClause ] [ord
erClause ] [ limitClause ]
hintCommand ::= < HINT stringLiteral { comma stringLiteral } >
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 type ] | identifier IN expr [ TYPE type ]
whereClause ::= WHERE expr
groupClause ::= GROUP BY valueList [ HAVING expr ]
orderClause ::= ORDER BY sortCriterion { comma sortCriterion }
sortCriterion ::= expr [ ASC | DESC ]
limitClause ::= LIMIT ( queryParam | integerLiteral )
expr ::= castExpr
castExpr ::= orExpr | left_paren type right_paren castExpr
orExpr ::= andExpr { OR andExpr }
andExpr ::= equalityExpr { AND equalityExpr }
equalityExpr ::= relationalExpr { ( = | <> | != ) relationalExpr }
relationalExpr ::= additiveExpr { ( < | <= | > | >= ) additiveExpr }
additiveExpr ::= multiplicativeExpr { (+ | -) multiplicativeExpr }
multiplicativeExpr ::= inExpr { (MOD | % | / | *) inExpr}
inExpr ::= unaryExpr { IN unaryExpr }
unaryExpr ::= { NOT | plus | dash } postfixExpr
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
      | aggregateExpr
      | queryParam
      | literal
      | left_paren query right_paren
      | regionPath
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 }
aggregateExpr ::= ( SUM | AVG ) left_paren [ DISTINCT ] expr right_paren
      | ( MIN | MAX ) left_paren expr right_paren
      | COUNT left_paren ( * | [ DISTINCT ] expr ) right_paren
queryParam ::= $ integerLiteral
regionPath ::= forward_slash regionName { forward_slash regionName }
regionName ::= nameCharacter { nameCharacter }
type ::= SHORT
      | LONG
      | INT
      | FLOAT
      | DOUBLE
      | CHAR
      | STRING
      | BOOLEAN
```

```
    | BYTE
    | OCTET
    | DATE
    | TIME
    | TIMESTAMP
    | SET < type >
    | COLLECTION < type >
    | LIST < type >
    | ARRAY < type >
    | ( DICTIONARY | MAP ) < type comma type >
    | identifier
identifier ::= letter { nameCharacter }
    | quotedIdentifier
quotedIdentifier ::= double_quote letter { nameCharacter } double_quote
literal ::= booleanLiteral
    | integerLiteral
    | longLiteral
    | doubleLiteral
    | floatLiteral
    | charLiteral
    | stringLiteral
    | dateLiteral
    | timeLiteral
    | timestampLiteral
    | NIL
    | 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
nameCharacter ::= 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 ::= '
double_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
  'They said, ''Hello''' value = They 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

VMware Tanzu GemFire supports comparison, logical, unary, arithmetic, 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:

Operator	Meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
=	equal to
!=	not equal to
<>	not equal to

Regarding equality and inequality operators:

- The equality and inequality operators have lower precedence than the other comparison operators.
- The equality and inequality operators can be used with null.
- Inequality queries return results for which the search field is UNDEFINED.
- 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. Tanzu GemFire supports the unary operator NOT. NOT is the negation operator, which changes the value of the operand to its opposite. For example, if an expression evaluates to TRUE, NOT changes it to FALSE. The operand must be a boolean.

Arithmetic Operators

Arithmetic operators operate on two values or expressions. Any of the expected arithmetic exceptions may result, such as overflow or a divide by zero. QueryInvocationTargetException will be thrown, and getCause() will state ArithmeticException.

The following are supported arithmetic operators:

Operator	Meaning
+	addition
-	subtraction
*	multiplication
/	division
%	modulus
MOD	modulus

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

This topic lists the words reserved for the VMware Tanzu GemFire query language.

Reserved Words

These case-insensitive words are reserved for the query language and may not be used as identifiers. The words followed with an asterisk (*) are not currently used by GemFire, but are reserved for future implementation.

abs*
all
and
andthen*
any*
array
as
asc
avg
bag*
boolean
by
byte
char
collection
count
date
declare*
define*
desc

dictionary
distinct
double
element
enum*
except*
exists*
false
first*
flatten*
float
for*
from
group
having
import
in
int
intersect*
interval*

is_defined
is_undefined
last*
like
limit
list
listtoset*
long
map
max
min
mod
nil
not
null
nvl
octet
or
order

orelse*
query*
select
set
short
some*
string
struct*
sum
time
timestamp
to_date
true
type
undefine*
undefined
union*
unique*
where

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

VMware Tanzu 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 fffffffff represents the fractional seconds (up to nine digits).

NIL

Equivalent alternative of NULL.

NULL

The same as null in Java.

UNDEFINED

A special literal, valid value for any data type, indicating that no value (not even NULL) has been designated for a given data item.

The Difference Between NULL and UNDEFINED

In OQL, as in Java, NULL is an assignable entity (an object) indicating “no value”.

In OQL, UNDEFINED is a type. There is no Java equivalent. In OQL search results, an UNDEFINED value can be returned in two cases:

- As the result of a search for a key or value that does not exist
- As the result of accessing an attribute of a null-valued attribute.

Searches for inequality return UNDEFINED values in their results.

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 for more information.

Query Evaulation 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 for more information.

Query Language Restrictions and Unsupported Features

This topic lists query language restrictions and unsupported features in VMware Tanzu GemFire.

At a high level, Tanzu GemFire does not support the following querying features:

- Indexes targeted for joins across more than one region are not supported
- Static method invocations are not supported. For example, the following query is invalid:

  ```
  SELECT DISTINCT * FROM /QueryRegion0 WHERE aDay = Day.Wednesday
  ```

- 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 the Geode model, attributes and methods are resolved at runtime during query execution. This means that OQL queries are not pre-validated for references to nonexistent methods and attributes. In addition, there are some specific limitations on partitioned region querying. For more information, see Partitioned Region Query Restrictions.

Advanced Querying

This topic links to VMware Tanzu GemFire advanced querying topics such as using query indexes, using query bind parameters, querying partitioned regions, and query debugging.

- Performance Considerations

  This topic covers considerations for improving query performance.

- Monitoring Low Memory When Querying

  The query monitoring feature prevents out-of-memory exceptions from occurring when you execute queries or create indexes.

- Managing Queries

  Manage the queries running in a cluster.

- Timeouts for Long-Running Queries

  Configure a timeout value for long running queries, such that they do not complete, and Tanzu GemFire throws an exception when a query runs for longer than the configured value.

- Using Query Bind Parameters

  Using query bind parameters in Tanzu GemFire queries is similar to using prepared statements in SQL where parameters can be set during query execution. This allows you to build a query once and execute it multiple times by passing the query conditions during run time.

- Querying a Specific Member

  A query may return different results depending on the member on which it is executed.

- Querying Partitioned Regions

Tanzu 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 Tanzu 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.

- 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.

Querying a Specific Member

This topic explains how to query a specific member of a region in VMware Tanzu GemFire.

When a region is specified with both REPLICATE and REPLICATE_PROXY region shortcuts on the server, a query could return a different result depending on the member on which it is executed.

Here is an example to illustrate this:

```
gfsh>connect --locator "localhost[64570]"
Connecting to Locator at [host=localhost, port=64570] ..
Connecting to Manager at [host=localhost, port=20571] ..
Successfully connected to: [host=localhost, port=20571]

gfsh>describe region --name "portfolio"
Name            : portfolio
Data Policy     : replicate
Hosting Members : server-2

Non-Default Attributes Shared By Hosting Members

Type   | Name        | Value
------ | ----------- | ---------------
Region | data-policy | REPLICATE
       | size        | 10
       | scope       | distributed-ack

Name            : portfolio
Data Policy     : empty
Accessor Members : server-1

Non-Default Attributes Shared By Accessor Members

Type   | Name        | Value
------ | ----------- | ---------------
Region | data-policy | EMPTY
       | size        | 0
       | scope       | distributed-ack
```

In this example, if the query is executed on a member with REPLICATE_PROXY, the result will be 0, as it will be a member without hosting data. To cope with this issue, you can specify the --member option forcing the query to be run on a member hosting the data.

```
gfsh>query --query="select ID, status from /portfolio where ID < 2"
Result : true
Limit : 100
Rows  : 0

gfsh>query --query="select ID, status from /portfolio where ID < 2" --member="server-2"
Result : true
Limit  : 100
Rows   : 2

ID | status
--- | --------
0  | active
1  | inactive (edited)
```

Performance Considerations

This topic list considerations for improving query performance in VMware Tanzu GemFire.

Some general performance tips:

- Improve query performance whenever possible by creating indexes. See Tips and Guidelines for Using Indexes 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 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 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.
- 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.

Managing Queries

This topic links to the gfsh commands used to manage the queries running in VMware Tanzu GemFire clusters.

List Queries

Display all queries in a cluster sorted by elapsed execution time. See list queries.

Describe Queries

Display detailed information about a query. See describe query.

Cancel Queries

Cancel a query and return an exception to the requestor. See cancel query.

Monitoring Low Memory When Querying

This topic explains how to monitor for low memory when querying in VMware Tanzu GemFire.

The query monitoring feature prevents out-of-memory exceptions from occurring when you execute queries or create indexes.

You enable this feature when you set a critical-heap-percentage attribute for the resource-manager element in the cache.xml file or by using the cache.getResourceManager().setCriticalHeapPercentage(float heapPercentage) API. When this feature is enabled and heap memory usage exceeds the threshold due to running a query or creating an index, the resource manager throws an exception and cancels the running query or index creation.

You can explicitly deactivate this feature by setting the system property gemfire.cache.DISABLE_QUERY_MONITOR_FOR_LOW_MEMORY to true.

When system memory is low, as determined by the critical heap percentage threshold defined in the cache.xml file 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.

Timeouts for Long-Running Queries

This topic explains how to set timeout values for long-running queries in VMware Tanzu GemFire.

Tanzu GemFire can monitor and throw an exception when a query runs longer than a configured amount of time. This feature can be enabled by setting the gemfire.Cache.MAX_QUERY_EXECUTION_TIME system property which detects that the JVM has too little heap memory.

The default query timeout is five hours. Set a different amount of time, in milliseconds, by specifying the system variable gemfire.Cache.MAX_QUERY_EXECUTION_TIME. A value of -1 explicitly deactivates the timeout.

When enabled, a query that runs longer than the configured timeout will be canceled such that it does not finish, and GemFire throws a QueryExecutionTimeoutException.

Using Query Bind Parameters

This topic explains how to use query bind parameters in VMware Tanzu GemFire.

Using query bind parameters in Tanzu 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.

Query objects are thread-safe.

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 Query.execute in the 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

```
// specify the query string
String queryString = "SELECT DISTINCT * FROM /exampleRegion p WHERE p.id = $1 and p.status = $2";

QueryService queryService = cache.getQueryService();
Query query = queryService.newQuery(queryString);

// set query bind parameters
Object[] params = new Object[2];
params[0] = 2;
params[1] = "active";

// Execute the query locally. It returns the results set.
SelectResults results = (SelectResults) query.execute(params);

// use the results of the query; this example only looks at the size
 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.

Examples:

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
```

Get all the keys from the region passed as a Region object for $1 for those entries whose name attribute is equal to the value passed for $2:

```
SELECT e.key FROM ($1).entrySet e WHERE e.value.name=$2"
```

Querying Partitioned Regions

This topic explains how to query partitioned regions in VMware Tanzu GemFire.

Tanzu 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 Tanzu 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 Tanzu 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 for Query and Querying a Partitioned Region on a Single Node 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 for Query and Optimizing Queries on Data Partitioned by a Key or Field Value 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 colocated, and you need to use the need to use Query.execute(RegionFunctionContext context). See the Java API for Query and Performing an Equi-Join Query on Partitioned Regions for more details.
- Using ORDER BY on Partitioned Regions
- Querying a Partitioned Region on a Single Node
- Optimizing Queries on Data Partitioned by a Key or Field Value
- Performing an Equi-Join Query on Partitioned Regions
- Partitioned Region Query Restrictions

Using ORDER BY on Partitioned Regions

This topic explains how to use the ORDER BY clause on a partitioned region in VMware Tanzu GemFire.

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 query coordinator gathers all results.

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

This topic explains how to query a partitioned region on a single node in VMware Tanzu GemFire.

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;
    }
  }
}
```

2. 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 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 colocated in the same bucket.
 */
public class MyPartitionResolver implements PartitionResolver, Declarable {

   public Serializable getRoutingObject(EntryOperation operation) {
   return operation.getKey().getId();
}
```

3. 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)
          .setArguments(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

This topic explains how to optimize queries on data partitioned by a key or field value in VMware Tanzu GemFire.

You can improve query performance on data that is partitioned by a 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 example shows how to optimize a query that will 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. For more information about creating key indexes, see Creating Key Indexes.
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)
          .setArguments(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

This topic explains how to perform an equi-join query on partitioned regions in VMware Tanzu GemFire.

To perform equi-join operations on partitioned regions or partitioned regions and replicated regions, you must use the query.execute method and supply it with a function execution context. You must use Tanzu GemFire's FunctionService executor because join operations are not directly supported for partitioned regions without providing a function execution context.

For more information about partitioned region query limitations, see Partitioned Region Query Restrictions.

Example equi-join query:

```
SELECT DISTINCT * FROM /QueryRegion1 r1,
/QueryRegion2 r2 WHERE r1.ID = r2.ID
```

In this example QueryRegion2 is colocated 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" ))
   .setArguments(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, 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(..).setArguments()). Then you can handle the parameters inside the function on the server side using context.getArguments. It does not matter in 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.

Partitioned Region Query Restrictions

This topic lists partitioned region query restrictions in VMware Tanzu GemFire.

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 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 Colocate Data from Different Partitioned Regions for more information about 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 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.
- If a partition region (or a bucket) being queried has been destroyed, the query is reattempted on the new primary for the destroyed bucket (if it exists). After certain number of attempts, a QueryException is thrown if all buckets (calculated at the startup of the query) cannot be queried.

Query Debugging

This topic explains how to debug queries in VMware Tanzu GemFire. You can debug a specific query at the query level by adding the <trace> keyword before the query string that you want to debug.

Example:

```
<trace> select * from /exampleRegion
```

You can also write:

```
<TRACE> select * from /exampleRegion
```

Before the query is executed and once it completes, Tanzu GemFire will log messages in $GEMFIRE_DIR/system.log with the following information:

```
[info 2011/08/29 11:24:35.462 PDT CqServer <main> tid=0x1] Starting query: select * from /exampleRegion

[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 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; indexe
  sUsed(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; ind
  exesUsed(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; index
  esUsed(2):sampleIndex-2(Results: 100),sampleIndex-1(Results: 199) "select * from /test k where ID > 0 OR status='ac
  tive'"
  ```

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.

Working with Indexes

This topic describes how to work with indexes in VMware Tanzu GemFire, and provides links to documentation that explains working with indexes in greater detail.

The Tanzu GemFire 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 for 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.

- Creating, Listing, and Removing Indexes

  The Tanzu 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 Key Indexes

  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 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.

- Creating Hash Indexes

  Hash indexes are deprecated. Tanzu GemFire supports creating hash indexes for the purposes of performing equality-based queries.

- Creating Indexes on Map Fields (“Map Indexes”)

  To assist with the quick lookup of multiple values in a Map (or HashMap) type field, you can create an index (sometimes referred to as a “map index”) on specific (or all) keys in that field.

- Creating Multiple Indexes Concurrently

  In order to speed and promote efficiency when creating indexes, you can define multiple indexes and then create them all at once.

- Maintaining Indexes (Synchronously or Asynchronously) and Index Storage

  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.

- Using Query Index Hints

  You can use the hint keyword to allow Tanzu GemFire’s query engine to prefer certain indexes.

- Using Indexes on Single Region Queries

  Queries with one comparison operation may be improved with either a key or range index, depending on whether the attribute being compared is also the primary key.

- 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.

- Using Indexes with Overflow Regions

  You can use indexes when querying on overflow regions; however, there are caveats.

- 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.

- Index Samples

  This topic provides code samples for creating query indexes.

Tips and Guidelines for Using Indexes

This topic provides tips and guideline for using indexes with queries in VMware Tanzu GemFire.

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 for details.
- If you are creating multiple indexes on the same region, first define your indexes and then create the indexes all at once to avoid iterating over the region multiple times. See Creating Multiple Indexes at Once 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 Tanzu 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.
- Whenever possible, provide a hint to allow the query engine to prefer a specific index. See Using Query Index Hints

Creating, Listing, and Removing Indexes

This topic explains how to create, list, and remove indexes in VMware Tanzu GemFire.

The Tanzu 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 for more information.
- Deprecated. createHashIndex. Creates a hash index. See Creating Hash Indexes for more information.
- createDefinedIndexes. Creates multiple indexes that were previously defined using defineIndex. See Creating Multiple Indexes at Once for more information.

The following sections provide examples of index creation:

Using gfsh:

```
gfsh> create index --name=myIndex --expression=status --region=/exampleRegion
gfsh> create index --name=myKeyIndex --expression=id --region=/exampleRegion --type=key
```

For more examples, see Index Commands in *Basic GemFire gfsh Commands by Functional Area*.

Using Java API:

```
QueryService qs = cache.getQueryService();
qs.createIndex("myIndex", "status", "/exampleRegion");
qs.createKeyIndex("myKeyIndex", "id", "/exampleRegion");
```

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"/>
  ...
</region>
```

: If you do not specify the type of index, 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 gfsh:

```
gfsh> list indexes
gfsh> list indexes --with-stats
```

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
```

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 gfsh:

```
gfsh> destroy index
gfsh> destroy index --name=myIndex
gfsh> destroy index --region=/exampleRegion
```

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
```

Creating Key Indexes

This topic explains how to create a key index for queries in VMware Tanzu GemFire.

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 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.

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 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.

: Using a key-index with an explicit type="range" in the cache.xml will result in an

exception. Key indexes cannot 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=region1.Id --region=region1 --type=key
```

Using cache.xml:

```
<region name=exampleRegion>
  <region-attributes ...>
  </region-attributes>
  <index name="myKeyIndex" from-clause="/exampleRegion" expression="id" key-index="true" type="key"/>
  ...
</region>
```

If you do not specify the type of index, the type defaults to range.

Creating Hash Indexes

This topic explains how to create a hash index for queries in VMware Tanzu GemFire.

Hash indexes are deprecated.

Tanzu GemFire supports creating hash indexes for the purpose of performing equality-based queries.

Hash Index Performance

The performance of put operations and recovery time when using a hash index will be worse than a range index. Queries are expected to be slower due to the implementation of the hash index and the cost of recalculating the key on request. A hash index can improve the memory usage of the index. So, the trade-off of the hash index space savings must be weighed against the performance penalty it imposes. If memory usage is not a concern, a range index is recommended.

Consider the memory usage when an index contains string fields. Copies of the strings are included in the index. With hash indexes, indexed expressions are canonicalized and stored in the index as pointers to the objects residing in the region, thereby using less memory. Tests achieved as high as a 30% reduction in memory footprint, but the savings depend on the keys and data being used.

Performance Considerations

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.
- You cannot use hash indexes for queries with multiple iterators or nested collections.
- Using a hash index will degrade put operation performance and recovery time substantially. If memory is not a concern, use a range index instead of a hash index.

Examples of Creating a Hash Index

Hash indexes are deprecated.

Using the Java API:

```
QueryService qs = cache.getQueryService();
qs.createHashIndex("myHashIndex", "mktValue", "/exampleRegion");
```

Using gfsh:

```
gfsh> create index --name=myHashIndex --expression=mktValue --region=/exampleRegion --type=hash
```

Using cache.xml:

```
<region name=exampleRegion>
  <region-attributes ...>
  </region-attributes>
  <index name="myHashIndex" from-clause="/exampleRegion p" expression="p.mktValue" type="hash"/>
  ...
</region>
```

Creating Indexes on Map Fields (“Map Indexes”)

This topic explains how to create indexes on Map Fields in VMware Tanzu GemFire.

To assist with the quick lookup of multiple values in a Map (or HashMap) type field, you can create an index (or “map index”) on specific or all keys in that field.

For example, you could create a map index to support the following query:

```
SELECT * FROM /users u WHERE u.name['first'] = 'John' OR u.name['last'] = 'Smith'
```

The map index extends regular range indexes created on single key by maintaining indexes for other specified keys, or for all keys if * is used. The underlying structure of the map index can be thought of as a wrapper around all these indexes.

The following Java code samples provide examples of how to create a map index:

```
QueryService qs = cache.getQueryService();

//This will create indexes for keys 'PVTL' and 'VMW'
qs.createIndex("indexName", "p.positions['PVTL', 'VMW']", "/portfolio p");
```

```
QueryService qs = cache.getQueryService();

//This will create indexes for all keys
qs.createIndex("indexName", "p.positions[*]", "/portfolio p");
```

In gfsh, the equivalents are:

```
gfsh>create index --name="IndexName" --expression="p.positions['PVTL', 'VMW']" --region="/portfolio p" --type=range

gfsh>create index --name="IndexName" --expression="p.positions[*]" --region="/portfolio p" --type=range
```

To create or query a map index, you must use the bracket notation to list the map field keys you wish to index or query. For example: [*], ['keyX1','keyX2']. Note that using p.pos.get('keyX1') will not create or query the map index.

> You can still query against Map or HashMap fields without querying against a map index. For example, you can always create a regular range query on a single key in any Map or HashMap field. However, note that subsequent query lookups will be limited to a single key.
>
> Map indexes on all keys (when * is specified) are not used in queries where the map field is compared with null or when it is compared using !=, e.g. p.positions['company1'] = null or p.positions['company1'] != '3'. These types of queries will be executed without making use of the index, which could make them slower.

Creating Multiple Indexes Concurrently

This topic explains how to create multiple indexes at the same time in VMware Tanzu GemFire.

In order to speed and promote efficiency when creating indexes, you can define multiple indexes and then create them all at once.

Defining multiple indexes before creating them speeds up the index creation process by iterating over region entries only once.

You can define multiple indexes of different types at once by specifying the --type parameter at definition time.

To define multiple indexes, you can use gfsh or the Java API:

gfsh example:

```
gfsh> define index --name=myIndex1 --expression=exp1 --region=/exampleRegion

gfsh> define index --name=myIndex2 --expression="c.exp2" --region="/exampleRegion e, e.collection1 c"

gfsh> create defined indexes
```

If index creation fails, you may receive an error message in gfsh similar to the following:

```
gfsh>create defined indexes
Exception : org.apache.geode.cache.query.RegionNotFoundException ,
Message : Region ' /r3' not found: from  /r3Occurred on following members
1. india(s1:17866)<v1>:27809
```

Java API example:

```
Cache cache = new CacheFactory().create();
  QueryService queryService = cache.getQueryService();
  queryService.defineIndex("name1", "indexExpr1", "regionPath1");
  queryService.defineIndex("name2", "indexExpr2", "regionPath2");
  queryService.defineKeyIndex("name4", "indexExpr4", "regionPath2");
  List<Index> indexes = queryService.createDefinedIndexes();
```

If one or more index population fails, Tanzu GemFire collect the Exceptions and continues to populate the rest of the indexes. The collected Exceptions are stored in a Map of index names and exceptions that can be accessed through MultiIndexCreationException.

Index definitions are stored locally on the gfsh client. If you want to create a new set of indexes or if one or more of the index creations fail, you might want to clear the definitions stored by using clear defined indexes command. The defined indexes can be cleared by using the Java API:

```
queryService.clearDefinedIndexes();
```

Or by using gfsh:

```
gfsh> clear defined indexes
```

Maintaining Indexes (Synchronously or Asynchronously) and Index Storage

This topic explains index storage, and how to maintain indexes, either synchronously or asynchronously, in VMware Tanzu GemFire.

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.

Index Maintenance Behavior

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 RegionFactory.setIndexMaintenanceSynchronous in the Java API docs.

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 Tanzu 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.

Compact indexes only support creating 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 Query Index Hints

This topic explains how to use the hint keyword to allow VMware Tanzu GemFire's query engine to prefer certain indexes.

In cases where one index is hinted in a query, the query engine filters off the hinted index (if possible) and then iterates and filters from the resulting values.

Example:

```
<HINT 'IDIndex'> SELECT * FROM /Portfolios p WHERE p.ID > 10 AND p.owner = 'XYZ'
```

If multiple indexes are added as hints, then the query engine will try to use as many indexes as possible while giving a preference for the hinted indexes.

Example:

```
<HINT 'IDIndex', 'OwnerIndex'> SELECT * FROM /Portfolios p WHERE p.ID > 10 AND p.owner = 'XYZ' AND p.value < 100
```

Using Indexes on Single Region Queries

This topic explains how to use indexes on single-region queries in VMware Tanzu GemFire.

Queries with one comparison operation may be improved with either a key or range index, depending on whether the attribute being compared is also the primary key.

If pkid is the key in the /exampleRegion region, creating a key index on pkid is the best choice as a key index does not have maintenance overhead. If pkid is not the key, a range index on pkid should improve performance.

```
SELECT DISTINCT * FROM /exampleRegion portfolio WHERE portfolio.pkid = '123'
```

With multiple comparison operations, you can create a range 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 range 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 pos
itions.MktValue > 1
```

Using Indexes with Equi-Join Queries

This topic explains how to use indexes with equi-join queries in VMware Tanzu GemFire.

Equi-join queries are queries in which two regions are joined through an equality condition in the WHERE clause.

To use an index with an equi-join query:

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.

   > 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 additional, single-region queries in a query with an equi-join condition exist, create additional indexes for the single-region conditions only if you can 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
/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

This topic lists the requirements for using indexes when querying overflow regions in VMware Tanzu GemFire.

Requirements:

- 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 on or 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("indx1", "entries.value.getID", "/exampleRegion.entrySet() entries");
```

The following example indexes will work for overflow regions.

```
createIndex("pkidIndex", "p.pkid", "/Portfolios p");

createIndex("indx1", "ks.toString", "/portfolio.keySet() ks");
```

The same working examples in gfsh:

```
gfsh> create index -name="pkidIndex" --expression="p.pkid" --region="/Portfolios p"

gfsh> create index -name="indx1" --expression="ks.toString" --region="/portfolio.keySet() ks"
```

Using Indexes on Equi-Join Queries using Multiple Regions

This topic explains how to use indexes on equi-join queries using multiple regions in VMware Tanzu GemFire.

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), then 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

This topic provides code samples for creating query indexes in VMware Tanzu GemFire.

```
// Key index samples. The field does not 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.valules")

//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
```

Continuous Querying

This topic links to documentation about VMware Tanzu GemFire continuous querying (CQ). Continuous querying continuously returns events that match the queries that you set.

- 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.

- Implementing Continuous Querying

  Use continuous querying in your clients to receive continuous updates to queries run on the servers.

- Managing Continuous Querying

  This topic discusses CQ management options, CQ states, and retrieving initial result sets.

How Continuous Querying Works

This topic explains how continuous querying (CQ) in VMware Tanzu GemFire 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, unless the event is configured to be filtered. 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.

Query operations based on old and new entry values	New value does not satisfy the query	New value satisfies the query
Old value does not satisfy the query	no event	CREATE query operation
Old value does satisfies the query	DESTROY query operation	UPDATE query operation

You can use the query operation to decide what to do with the CQ event listener 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.

CQ Event Filter

Users can specify the following types of events to exclude when defining a new CQ:

- CREATE
- DESTROY
- INVALIDATE
- UPDATE

Example code:

```
CqAttributesFactory cqAttributesFactory = new CqAttributesFactory();

cqAttributesFactory.setExcludedEvents(Collections.singleton(ExcludedEvent.DESTROY));

qs.newCq("select * from /Region ", cqAttributesFactory.create()).execute();
```

ExcludedEvent is an enum that represents the category of events that will be filtered out. Events are associated with an Operation, and each Operation is related to an OpType. There are many Operations per OpType.

Table 1: Operations and Related OpTypes

Operations	Related OpTypes
Operation.CREATE	CREATE
Operation.DESTROY	DESTROY
Operation.EVICT_DESTROY	DESTROY
Operation.EXPIRE_DESTROY	DESTROY
Operation.EXPIRE_INVALIDATE	INVALIDATE
Operation.EXPIRE_LOCAL_DESTROY	DESTROY
Operation.EXPIRE_LOCAL_INVALIDATE	INVALIDATE
Operation.INVALIDATE	INVALIDATE
Operation.LOCAL_DESTROY	DESTROY
Operation.LOCAL_INVALIDATE	INVALIDATE
Operation.LOCAL_LOAD_CREATE	CREATE
Operation.LOCAL_LOAD_UPDATE	UPDATE
Operation.PUTALL_CREATE	CREATE
Operation.PUTALL_UPDATE	UPDATE
Operation.PUT_IF_ABSENT	CREATE
Operation.REMOVE	DESTROY
Operation.REMOVEALL_DESTROY	DESTROY
Operation.REPLACE	UPDATE
Operation.UPDATE	UPDATE

When a CREATE event fails due to node failure, GemFire will automatically retry the event. The CREATE event might be converted to an UPDATE event on the retried node,

because it has already processed the CREATE event. This can cause the CQ event listener to either miss or incorrectly trigger an extra event when the CQ Event Filter is set to exclude CREATE or UPDATE events.

Region Type Restrictions for CQs

You can only create CQs on replicated or partitioned regions. If you attempt to create a CQ on a non-replicated or non-partitioned region, you will receive the following error message:

```
The region <region name> specified in CQ creation is neither replicated nor partitioned; only replicated or partitioned regions are allowed in CQ creation.
```

In addition, you cannot create a CQ on a replicated region with eviction setting of local-destroy since this eviction setting changes the region's data policy. If you attempt to create a CQ on this kind of region, you will receive the following error message:

```
CQ is not supported for replicated region: <region name> with eviction action: LOCAL_DESTROY
```

See also Configure Distributed, Replicated, and Preloaded Regions for potential issues with setting local-destroy eviction on replicated regions.

Implementing Continuous Querying

This topic explains how to implement continuous querying (CQ) in VMware Tanzu GemFire.

Use continuous querying in your clients to receive continuous updates to queries run on the servers.

Before you begin, you should be familiar with Querying 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 such as "/tradeOrder.name" or "(SELECT * from /tradeOrder).size".
   - The CQ query cannot use:
     - Cross region joins
     - Drill-downs into nested collections
     - DISTINCT
     - Projections
     - Bind parameters
     - ORDER BY

        - GROUP BY
        - Aggregate Functions (MIN, MAX, AVG, SUM, COUNT)
    - The CQ query must be created on a partitioned or replicated region. See Region Type Restrictions for CQs.

The basic syntax for the CQ query is:

```
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 org.apache.geode.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.

> 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)
  {
    // org.apache.geode.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 or disconnected from the servers. 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)
  {
    // org.apache.geode.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:
    1. Create a CqAttributesFactory and use it to set your CqListeners and CqStatusListener.
    2. Pass the attributes factory and the CQ query and its unique name to the QueryService to create a new CqQuery.
    3. 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.
    4. 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();
  for (Object o : sResults) {
      Struct s = (Struct) o;
      TradeOrder to = (TradeOrder) s.get("value");
      System.out.println("Intial result includes: " + to);
  }
}
  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.

Managing Continuous Querying

This topic discusses VMware Tanzu GemFire continuous querying (CQ) management options, CQ states, and retrieving initial result sets.

Using CQs from a RegionService Instance

If you are running durable client queues from the RegionService instance, 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.

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	QueryService.newCq	N/A
Execute a CQ	CqQuery.execute and CqQuery.executeWithInitialResults	QueryService.executeCqs
Stop a CQ	CqQuery.stop	QueryService.stopCqs
Close a CQ	CqQuery.close	QueryService.closeCqs
Access a CQ	CqEvent.getCq and QueryService.getCq	QueryService.getCq
Modify CQ Listeners	CqQuery.getCqAttributesMutator	N/A
Access CQ Runtime Statistics	CqQuery.getStatistics	QueryService.getCqStatistics
Get all durable CQs registered on the server	N/A	QueryService.getAllDurableCqsFromServer

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.
- List all durable CQs for a given durable client ID. See list.
- Show the subscription event queue size for a given durable client ID. See show subscription-queue-size.

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.

Transactions

This topic links to documentation about transactions in VMware Tanzu GemFire.

Tanzu GemFire offers an API for client applications that do transactional work. Tanzu GemFire implements optimistic transactions, with the familiar begin, commit, and rollback methods that implement the same operations as in relational database transactions methods.

- Adherence to ACID Promises

  This section explains the ways in which Tanzu GemFire's implementation of optimistic transactions provides ACID semantics.

- Code Examples

  An application-based transaction and a transaction embedded in a function provide examples to model.

- Design Considerations

  Designs that extend beyond the basics introduce other considerations. This section identifies and discusses how transactions interact with other aspects of the system.

Adherence to ACID Promises

This topic introduces VMware Tanzu GemFire transactions and discusses adherence to Atomicity-Consistency-Isolation-Durability (ACID).

Atomicity-Consistency-Isolation-Durability

Tanzu GemFire offers an API for client applications that do transactional work. Tanzu GemFire implements optimistic transactions, choosing the much higher transaction performance they offer over the slow, locking methods of a traditional relational database.

Optimistic transaction semantics are not identical to the Atomicity-Consistency-Isolation-Durability (ACID) semantics of a traditional relational database.

Atomicity

Atomicity is “all or nothing” behavior: a transaction completes successfully only when all of the operations it contains complete successfully. If problems occur during a transaction, perhaps due to other transactions with overlapping changes, the transaction cannot successfully complete until the problems are resolved.

Optimistic transactions provide atomicity and realize speed by using a reservation system, instead of using the traditional relational database technique of a two-phase locking of rows. The reservation prevents other, intersecting transactions from completing, allowing the commit to check for conflicts and to reserve resources in an all-or-nothing fashion prior to making changes to the data. After all changes have been made, locally and remotely, the reservation is released. With the reservation system, an intersecting transaction is simply discarded. The serialization of obtaining locks is avoided.

Consistency

Consistency requires that data written within a transaction must observe the key and value constraints established for the affected region. Note that validity of the transaction is the responsibility of the application.

Isolation

Isolation is the level at which transactional state is visible to system components. Tanzu GemFire transactions have repeatable read isolation. Once the committed value is read for a given key, it always returns that same value. If a write within a transaction deletes a value for a key that has already been read, subsequent reads return the transactional reference.

The default configuration isolates transactions at the process thread level. While a transaction is in progress, its changes are visible only inside the thread that is running the transaction. Other threads within that same process and threads in other processes cannot see changes until after the commit operation begins. After beginning the commit, the changes are visible in the cache, but other threads that access the changing data might see partial results of the transaction, leading to a dirty read. See Changing the Handling of Dirty Reads for how to change the default behavior.

Durability

Relational databases provide durability by using disk storage for recovery and transaction logging. Tanzu GemFire is optimized for performance and does not support on-disk durability for transactions.

See Allowing Transactions to Work on Persistent Regions for how to allow a transaction that operates on a persistent region in a non-durable way.

Code Examples

This topic provides code fragments for an example of an application running a transaction directly and an example of an application invoking a function that contains a transaction in VMware Tanzu GemFire.

An application can run a transaction directly or invoke a function which contains a transaction. This section illustrates these two use cases with code fragments that demonstrate the proper way to program a transaction.

An expected use case operates on two regions within a transaction. For performance purposes the Tanzu GemFire transaction implementation requires that region entries of partitioned regions be colocated. See Custom-Partitioning and Colocating Data for details on how to colocate region entries.

Transaction within an Application

An application/client uses the CacheTransactionManager API. This most basic code fragment shows the structure of a transaction, with its begin to start the transaction, commit to end the transaction, and handling of exceptions that these methods may throw.

```
CacheTransactionManager txManager =
  cache.getCacheTransactionManager();

try {
  txManager.begin();
  // ... do transactional, region operations
  txManager.commit();
} catch (CommitConflictException conflict) {
  // ... do necessary work for a transaction that failed on commit
} finally {
  // All other exceptions will be handled by the caller.
  // Examples of some exceptions: the data is not colocated, a rebalance
  // interfered with the transaction, or the server is gone.
  // Any exception thrown by a method other than commit() needs
  // to do a rollback to avoid leaking the transaction state.
  if(txManager.exists()) {
    txManager.rollback();
  }
}
```

More details of a transaction appear in this next application/client code fragment example. In this typical transaction, the put operations must be atomic and two regions are involved.

In this transaction, a customer's purchase is recorded. The cash region contains each customer's cash balance available for making trades. The trades region records each customer's balance spent on trades.

If there is a conflict upon commit of the transaction, an exception is thrown, and this example tries again.

```
// inputs needed for this transaction; shown as variables for simplicity
final String customer = "Customer1";
final Integer purchase = 1000;

// region set up shown to promote understanding
Cache cache = new CacheFactory().create();
Pool pool = PoolManager.createFactory()
  .addLocator("localhost", LOCATOR_PORT)
  .create("pool-name");
Region<String, Integer> cash =
  cache.createClientRegionFactory(ClientRegionShortcut.PROXY)
  .setPoolName(pool.getName())
  .create("cash");
Region<String, Integer> trades =
  cache.createClientRegionFactory(ClientRegionShortcut.PROXY)
  .setPoolName(pool.getName())
  .create("trades");

// transaction code
CacheTransactionManager txManager = cache.getCacheTransactionManager();
boolean retryTransaction = false;
do {
  try {
    txManager.begin();

    // Subtract out the cost of the trade for this customer's balance
    Integer cashBalance = cash.get(customer);
    Integer newBalance = (cashBalance != null ? cashBalance : 0) - purchase;
    cash.put(customer, newBalance);

    // Add in the cost of the trade for this customer
    Integer tradeBalance = trades.get(customer);
    newBalance = (tradeBalance != null ? tradeBalance : 0) + purchase;
    trades.put(customer, newBalance);

    txManager.commit();
    retryTransaction = false;
  }
  catch (CommitConflictException conflict) {
    // entry value changed causing a conflict for this customer, so try again
    retryTransaction = true;
  } finally {
    // All other exceptions will be handled by the caller.
    // Any exception thrown by a method other than commit() needs
    // to do a rollback to avoid leaking the transaction state.
    if(txManager.exists()) {
      txManager.rollback();
    }
  }
} while (retryTransaction);
```

Design transactions such that any get operations are within the transaction. This causes those entries to be part of the transactional state, which is desired such that intersecting transactions can be detected and signal commit conflicts.

Transaction within a Function

A transaction may be embedded in a function. The application invokes the function, and the function contains the transaction that does the begin, the region operations, and the commit or rollback.

This use of a function can have performance benefits. The performance benefit results from both the function and the region data residing on servers. As the function invokes region operations, those operations on region entries stay on the server, so there is no network round trip time to do get or put operations on region data.

This function example accomplishes atomic updates on a single region representing the quantity of products available in inventory. Doing this in a transaction prevents double allocating inventory for two orders placed simultaneously.

```
/**
 * Atomically reduce inventory quantity
 */
public class TransactionalFunction extends Function {

  /**
   * Returns true if the function had the requested quantity of
   * inventory and successfully completed the transaction to
   * record the reduced inventory that fulfills the order.
   */
  @Override
  public void execute(FunctionContext context) {
    RegionFunctionContext rfc = (RegionFunctionContext) context;
    Region<ProductId, Integer> inventoryRegion = rfc.getDataSet();

    CacheTransactionManager
      txManager = context.getCache().getCacheTransactionManager();

    // single argument will be a ProductId and a quantity
    ProductRequest request = (ProductRequest) rfc.getArguments();
    ProductId productRequested = request.getProductId();
    Integer qtyRequested = request.getQuantity();

    boolean success = false;

    do {
      boolean commitConflict = false;
      try {
        txManager.begin();

        Integer qtyAvailable = inventoryRegion.get(productRequested);
        if (qtyAvailable >= qtyRequested) {
          // enough inventory is available, so process request
          Integer remaining = qtyAvailable - qtyRequested;
          inventoryRegion.put(productRequested, remaining);
          txManager.commit();
          success = true;
        }

      } catch (CommitConflictException conflict) {
        // retry transaction, as another request on this same key succeeded,
        // so this transaction attempt failed
        commitConflict = true;
      } finally {
        // All other exceptions will be handled by the caller; however,
        // any exception thrown by a method other than commit() needs
        // to do a rollback to avoid leaking the transaction state.
        if(txManager.exists()) {
          txManager.rollback();
        }
      }

    } while (commitConflict);

    context.getResultSender().lastResult(success);
  }

  @Override
  public String getId() {
    return "TxFunction";
  }

  /**
```

```
   * Returning true causes this function to execute on the server
   * that holds the primary bucket for the given key. It can save a
   * network hop from the secondary to the primary.
   */
  @Override
  public boolean optimizeForWrite() {
    return true;
  }
}
```

The application-side details on function implementation are not covered in this example. The application sets up the function context and the argument. See the section on Function Execution for details on functions.

The function implementation needs to catch the commit conflict exception such that it can retry the entire transaction. The exception only occurs if another request for the same product intersected with this one, and that other request's transaction committed first.

The optimizeForWrite method is defined to cause the system to execute the function on the server that holds the primary bucket for the given key. It can save a network hop from the secondary to the primary.

Note that the variable qtyAvailable is a reference, because the Region.get operation returns a reference within this server-side code. Read Region Operations Return References for details and how to work around the implications of a reference as a return value when working with server code.

Design Considerations

This topic discusses how transactions interact with other VMware Tanzu GemFire features.

Designs that incorporate more complex features introduce further considerations:

- Colocate Partitioned Regions
- Region Operations Return References
- First Operation with Mixed Region Types
- Allowing Transactions to Work on Persistent Regions
- Mixing Transactions with Queries and Indexes
- Mixing Transactions with Eviction
- Mixing Transactions with Expiration
- Mixing Transactions with Non-transactional Operations
- Changing the Handling of Dirty Reads

Colocate Partitioned Regions

For performance, transactions that operate on more than one partitioned region require that those partitioned regions colocate their entries. Colocate Data from Different Partitioned Regions describes how to colocate entries.

Region Operations Return References

For performance, server-invoked region operations return references to region entries. Any assignment to that reference changes the entry within the region. This subverts the system's ability to maintain consistency and the callback chain for handlers such as cache writers and cache loaders.

Changing an entry using a reference from within a transaction executing on a server has the same consistency issues, but is even worse, as the change will not be seen as part of the transactional state.

There are two ways to work with a reference: make a copy, or configure the system to return copies instead of references. There is a performance penalty to having the system return copies. Both ways are detailed in Copy on Read Behavior.

First Operation with Mixed Region Types

When more than one region participates in a transaction, and there is at least one partitioned and at least one replicated region, the code must do its first operation on the partitioned region to avoid a TransactionDataNotColocatedException. Write the transaction to do its first operation on a partitioned region, even if the operation will be spurious.

Allowing Transactions to Work on Persistent Regions

Tanzu GemFire's implementation of atomic transactions prohibits regions with persistence from participating in transactions. The invocation of a persistent region operation within a transaction throws an UnsupportedOperationException with an associated message of

```
Operations on persist-backup regions are not allowed because this thread
has an active transaction
```

An application that wishes to allow operations on a persistent region during a transaction can set this system property:

-Dgemfire.ALLOW_PERSISTENT_TRANSACTIONS=true

Setting this system property eliminates the exception. It does not change the fact that atomicity is not enforced for disk writes that occur with the commit of a transaction. A server crash during the commit may succeed in some, but not all of the disk writes.

Mixing Transactions with Queries and Indexes

Queries and query results reflect region state, and not any state or changes that occur within a transaction. Likewise, the contents and updates to an index do not intersect with any changes made within a transaction. Therefore, do not mix transactions with queries or indexed regions.

Mixing Transactions with Eviction

LRU eviction and transactions work well together. Any eviction operation on a region entry that is operated on from within a transaction is deferred until the transaction is committed. Further, because any entry touched by the transaction has had its LRU clock reset, eviction is not likely to choose those entries as victims immediately after the commit.

Mixing Transactions with Expiration

A transaction deactivates expiration on any region entries affected by the transaction.

Mixing Transactions with Non-transactional Operations

For best performance, non-transactional operations do not acquire the exclusive locks used to check for conflicts in a transaction. A transaction operating on the same data as a non-transactional actor is unable

to detect the conflict caused by a non-transactional operation.

If using transactions, an application should adopt the policy of designating certain regions or sets of entries exclusively for transactional puts, updates, and deletions, so transactional entries will not be modified by non-transactional operations.

If other, non-transactional sources update the keys the transaction is modifying, the changes may intermingle with the 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.

Changing the Handling of Dirty Reads

An application requiring a strict, but slower isolation model, such that dirty reads of transitional states are not allowed, should set a property and encapsulate read operations within the transaction. Configure this strict isolation model with the property:

```
-Dgemfire.detectReadConflicts=true
```

This property causes read operations to succeed only when they read a consistent pre- or post-transactional state. If not consistent, Tanzu GemFire throws a CommitConflictException.

Function Execution

This topic links to documentation that explains function execution in VMware Tanzu GemFire.

A function is a body of code that resides on a server and that an application can invoke from a client or from another server without the need to send the function code itself. The caller can direct a data-dependent function to operate on a particular dataset, or can direct a data-independent function to operate on a particular server, member, or member group.

The function execution service provides solutions for a variety of use cases, including:

- An application needs to perform an operation on the data associated with a key. A registered server-side function can retrieve the data, operate on it, and put it back, with all processing performed locally to the server.
- An application needs to initialize some of its components once on each server, which might be used later by executed functions.
- A third-party service, such as a messaging service, requires initialization and startup.
- Any arbitrary aggregation operation requires iteration over local data sets that can be done more efficiently through a single call to the cache server.
- An external resource needs provisioning that can be done by executing a function on a server.

Check the following sections for more information:

- How Function Execution Works
- Executing a Function in VMware Tanzu GemFire

How Function Execution Works

This topic explains how function execution works in VMware Tanzu GemFire.

Where Functions Are Executed

You can execute data-independent functions or data-dependent functions in Tanzu GemFire in the following places:

For Data-independent Functions

- On a specific member or members: Execute the function within a peer-to-peer cluster, specifying the member or members where you want to run the function by using FunctionService methods onMember() and onMembers().
- On a specific server or set of servers: If you are connected to a cluster 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 using the default connection pool. For data-independent functions on client/server architectures, a client invokes FunctionService methods onServer() or onServers(). (See How Client/Server Connections Work for details regarding pool connections.)
- On member groups or on a single member within each member group: You can organize members into logical member groups. (See Configuring and Running a Cluster for more information about using member groups.) 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.

For Data-dependent Functions

- On a region: If you are executing a data-dependent function, specify a region and, optionally, a set of keys on which to run the function. The method FunctionService.onRegion() directs a data-dependent function to execute on a specific region.

See the org.apache.geode.cache.execute.FunctionService Java API documentation for more details.

How Functions Are Executed

The following things occur when executing a function:

1. For security-enabled clusters, prior to executing the function, a check is made to see that caller is authorized to execute the function. The required permissions for authorization are provided by the function's Function.getRequiredPermissions() method. See Authorization of Function Execution for a discussion of this method.
2. Given successful authorization, Tanzu GemFire invokes the function on all members where it needs to run. The locations are determined by the FunctionService on* method calls, region configuration, and any filters.
3. If the function has results, they are returned to the addResult method call in a ResultCollector object.
4. 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 Tanzu GemFire automatically retries a function if it does not execute successfully. You must code and configure the function to be highly available, and the calling application must invoke the function using the results collector getResult method.

When a failure (such as an execution error or member crash while executing) occurs, the system responds by:

1. Waiting for all calls to return

2. Setting a boolean indicating a re-execution
3. Calling the result collector's clearResults method
4. Executing the function

For client regions, the system retries the execution according to org.apache.geode.cache.client.Pool retryAttempts. If the function fails to run every time, the final exception is returned to the getResult method.

The default number of retries is the total number of servers present. For member calls, if a function fails then the system will retry on each server only once or until no data remains in the system for the function to operate on. If the function fails on every server, then an exception will be returned to the user.

Function Execution Scenarios

This figure shows the sequence of events for a data-independent function invoked from a client on all available servers.

The client contacts a locator to obtain host and port identifiers for each server in the cluster and issues calls to each server. As the instigator of the calls, the client also receives the call results.

This figure shows the sequence of events for a data-independent function executed against members in a peer-to-peer cluster.

You can think of onMembers() as the peer-to-peer counterpart of a client-server call to onServers(). Because it is called from a peer of other members in the cluster, an onMembers() function invocation has access to detailed metadata and does not require the services of a locator. The caller invokes the function on itself, if appropriate, as well as other members in the cluster and collects the results of all of the function executions.

Data-dependent Function on a Region shows a data-dependent function run on a region.

Figure: Data-dependent Function on a Region

An onRegion() call requires more detailed metadata than a locator provides in its host:port identifier. This diagram shows the path followed when the client lacks detailed metadata regarding target locations, as on the first call or when previously obtained metadata is no longer up to date.

The first time a client invokes a function to be executed on a particular region of a cluster, the client's knowledge of target locations is limited to the host and port information provided by the locator. Given only this limited information, the client sends its execution request to whichever server is next in line to be called according to the pool allocation algorithm. Because it is a participant in the cluster, that server has access to detailed metadata and can dispatch the function call to the appropriate target locations. When the server returns results to the client, it sets a flag indicating whether a request to a different server would have provided a more direct path to the intended target. To improve efficiency, the client requests a copy of the metadata. With additional details regarding the bucket layout for the region, the client can act as its own dispatcher on subsequent calls and identify multiple targets for itself, eliminating at least one hop.

After it has obtained current metadata, the client can act as its own dispatcher on subsequent calls, identifying multiple targets for itself and eliminating one hop, as shown in Data-dependent function after obtaining current metadata.

Figure: Data-dependent function after obtaining current metadata

Data-dependent Function on a Region with Keys shows the same data-dependent function with the added specification of a set of keys on which to run.

Figure: Data-dependent Function on a Region with Keys

Servers that do not hold any keys are left out of the function execution.

Peer-to-peer Data-dependent Function shows a peer-to-peer data-dependent call.

Figure: Peer-to-peer Data-dependent Function

The caller is a member of the cluster, not an external client, so the function runs in the caller's cluster. Note the similarities between this diagram and the preceding figure (Data-dependent Function on a Region with Keys), which shows a client-server model where the client has up-to-date metadata regarding target locations within the cluster.

Client-server system with Up-to-date Target Metadata demonstrates a sequence of steps in a call to a highly available function in a client-server system in which the client has up-to-date metadata regarding target locations.

Figure: Client-server system with Up-to-date Target Metadata

In this example, three primary keys (X, Y, Z) and their secondary copies (X', Y', Z') are distributed among three servers. Because optimizeForWrite is true, the system first attempts to invoke the function where the primary keys reside: Server 1 and Server 2. Suppose, however, that Server 2 is off-line, so the call targeted for key Y fails. Because isHA is set to true, the call is retried on Server 1 (which succeeded the first time, so likely will do so again) and Server 3, where key Y' resides. This time, the function call returns

successfully. Calls to highly available functions retry until they obtain a successful result or they reach a retry limit.

Executing a Function in Tanzu GemFire

This topic explains how to execute a function in VMware Tanzu GemFire.

Before starting this procedure, define your members and regions 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 for details. Alternatively, you can write the XML or application code to register the function. See Register the Function Programmatically 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 in the org.apache.geode.cache.execute package.

Code the methods you need for the function. These steps do not have to be done in this order.

- Implement getId to return a unique name for your function. You can use this name to access the function through the FunctionService API.
- For high availability:
    1. Code isHa to return true to indicate to Tanzu GemFire that it can re-execute your function after one or more members fails
    2. Code your function to return a result
    3. Code hasResult to return true
- 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.
- If the function will be executed on a region, implement 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. optimizeForWrite returns false by default.
- If the function should be run with an authorization level other than the default of DATA:WRITE, implement an override of the Function.getRequiredPermissions() method. See Authorization of Function Execution for details on this method.
- Code the execute method to perform the work of the function.
    1. Make execute thread safe to accommodate simultaneous invocations.
    2. 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.

> 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.

3. 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.

     > When you use PartitionRegionHelper.getLocalDataForContext, putIfAbsent may not return expected results if you are working on local data set instead of the region.

4. To propagate an error condition or exception back to the caller of the function, throw a FunctionException from the execute method. Tanzu GemFire transmits the exception back to the caller as if it had been thrown on the calling side. See the Java API documentation for FunctionException for more information.

Example function code:

```
import java.io.Serializable;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

import org.apache.geode.cache.execute.Function;
import org.apache.geode.cache.execute.FunctionContext;
import org.apache.geode.cache.execute.FunctionException;
import org.apache.geode.cache.execute.RegionFunctionContext;
import org.apache.geode.cache.partition.PartitionRegionHelper;

public class MultiGetFunction implements Function {

  public void execute(FunctionContext fc) {
    if(! (fc instanceof RegionFunctionContext)){
      throw new FunctionException("This is a data aware function, and has
 to be called using FunctionService.onRegion.");
    }
    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++)
    {
      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();
  }
}
```

> Ensure that the result being sent is an immutable object. If unsure, use CopyHelper.copy(Object) to create an immutable copy to send as the function result.

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 cluster 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.

: When deploying a JAR, the name of the JAR cannot contain any of the following characters: *, !, #, @, ?.

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 VMware Tanzu GemFire Members 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);
```

> 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 section assumes that you have 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 cluster 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 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 setArguments. 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.

> 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)
    .setArguments(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();
```

Tanzu 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:
    1. addResult and endResults are called by Tanzu GemFire when results arrive from the Function instance SendResults methods
    2. 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:
    1. Code the ResultCollector clearResults method to remove any partial results data. This readies the instance for a clean function re-execution.
    2. When you invoke the function, call the result collector getResult 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)
        .setArguments(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.

Distributed Lock Service

VMware Tanzu GemFire is a distributed system that uses locking mechanisms to coordinate certain operations. GemFire's Distributed Lock Service (DLS) provides general-purpose distributed locking across a GemFire cluster. The DLS is available only to peer members, which are members participating as full data-stores in the distributed system.

Client applications using the client cache cannot access or participate in the DLS. This limitation ensures proper lock coordination and prevents network inconsistencies.

There are no configuration options to allow DLS access from client applications. If client-side locking is required, you should implement an application-level coordination mechanism or use server-side functions that interact with the DLS on behalf of the client.

Creating the Distributed Lock Service

To use the DLS in an application, you first need to create an instance of the service. This service is identified by a distributed name that all participating members must agree upon.

Example:

```
DistributedLockService lockService =
    DistributedLockService.create("MyLockService", cache.getDistributedSystem());
```

Acquiring and Releasing Locks

Locks in DLS are identified by a string-based name. When an application acquires a lock, other threads attempting to acquire the same lock are blocked until the lock owner releases it, or the lock is forcibly released due to a failure.

This example locks the LockName resource using the specified wait time and lease time. If the method returns false, the lock could not be acquired before the waitTimeMillis elapsed. The caller should then verify whether the lock was successfully obtained.

```
boolean gotLock = lockService.lock("LockName", waitTimeMillis, leaseTimeMillis);
```

Where:

waitTime is how long the caller will wait to acquire the lock. A value of -1 means wait indefinitely.

leaseTime is how long the lock will be held before it is released automatically. A value of -1 means hold indefinitely.

Example:

```
lockService.unlock("LockName");
```

This releases the lock on the LockName resource.

Lock Grantor

A unique concept in GemFire's DLS implementation is the lock grantor. The lock grantor is the designated member in the distributed system responsible for managing lock ownership and arbitration. By default, any member can serve as the lock grantor. However, applications can explicitly request that the current member become the grantor to optimize performance, especially when lock access is centralized to a specific node. When the lock grantor is colocated with the code requesting locks, lock operations become local, eliminating the overhead of network communication and enabling near in-memory speeds.

Example:

```
lockService.becomeLockGrantor();
```

After invoking becomeLockGrantor(), the calling member becomes the lock grantor. Other members in the cluster must contact this member to acquire locks.

> If lock requests are evenly distributed across multiple members, explicitly becoming the grantor may not provide any benefit. Continuously relocating the lock grantor can incur unnecessary costs, so it should only be done when there is a clear performance improvement.

Error Handling

Always wrap lock operations in appropriate try-finally blocks to ensure locks are released, even when exceptions occur.

Example:

```
boolean locked = false;
try {
   // true if the lock was acquired, false if the timeout
   // waitTimeMillis passed without acquiring the lock.
   if(lockService.lock("LockName", 5000, 10000)){
      // Perform critical section logic
   }else{
       // Didn't get the lock in the time out
   }
} catch (Exception e) {
   // Handle acquisition failure
   log.error("Failed to acquire lock", e);
} finally {
   if (lockService.isHeldByCurrentThread("LockName") {
       try {
           lockService.unlock("LockName");
       } catch (Exception unlockEx) {
           log.warn("Failed to release lock", unlockEx);
       }
   }
}
```

Deadlock Handling

While GemFire does not provide built-in deadlock detection in the DLS, you can reduce deadlock risk by following best practices:

1. Consistent lock order: Acquire locks in the same order across all threads and members.
2. Avoid nested locks: Nested locks increase the risk of deadlocks, so they should be avoided when possible.
3. Use timeouts: Implementing timeouts ensures that a thread doesn't wait forever to acquire a lock, breaking potential deadlock cycles by allowing the lock request to fail after a set period.
4. Log lock activity: By logging lock acquisition and release, you can audit the system and debug any issues, which is particularly useful in identifying deadlocks or other locking problems.
5. External coordination: For more complex lock management or explicit deadlock detection, use an external system, such as a coordinator or state machine, to handle the lock orchestration.

Lock Acquisition Best Practices

Minimize Lock Scope: Keep the code in the critical section as short and fast as possible.

Avoid Lock Contention: Partition lock names or use application-level sharding to reduce high-contention hot spots.

Monitor Lock Usage: Consider logging or using metrics to track how often locks are held and how long they're waited on.

Fail Fast: Use finite wait times over indefinite waits (-1) unless absolutely required.

Graceful Degradation: Design fallback logic to handle cases when a lock cannot be acquired.

Developing REST Applications for Tanzu GemFire

This topic provides links to background documentation and instructions for programming REST applications with VMware Tanzu GemFire. Tanzu GemFire REST APIs allow you to access region data, queries, and functions in your Tanzu GemFire deployment in wide variety of programming languages.

> This documentation covers the v1 release of Tanzu GemFire REST APIs for developing applications.

- Tanzu GemFire REST API Overview

  By using the Tanzu GemFire REST application interface, you can immediately access Tanzu GemFire's data management capabilities in languages other than the natively supported Java language.

- Prerequisites and Limitations for Writing REST Applications

  Before development, understand the prerequisites and limitations of the current REST implementation in Tanzu GemFire.

- Setup and Configuration

  The VMware Tanzu GemFire developer REST interface runs as an embedded HTTP or HTTPS service (Jetty server) within a Tanzu GemFire data node.

- Using the Swagger UI to Browse REST APIs

  VMware Tanzu GemFire Developer REST APIs are integrated with the Swagger™ framework. This framework provides a browser-based test client that allows you to visualize and try out Tanzu GemFire REST APIs.

- Developing REST Applications

  This section provides guidelines on writing REST client applications for Tanzu GemFire.

- Sample REST Applications

  This section provides examples that illustrate how multiple clients, both REST and native, can access the same Tanzu GemFire region data.

- Troubleshooting and FAQ

  This section provides troubleshooting guidance and frequently asked questions about Tanzu GemFire Developer REST APIs.

- VMware Tanzu GemFire REST API Reference

  This section summarizes all available VMware Tanzu GemFire REST API resources and endpoints.

GemFire REST API Overview

The VMware Tanzu GemFire REST application interface allows you to immediately access Tanzu GemFire's data management capabilities in languages other than the natively supported Java language.

You can write REST-enabled client applications for Tanzu GemFire in a variety of languages that use the open and standard HTTP protocol. Examples of these include Ruby, Python, and JavaScript, as well as languages that are already supported such as Java.

When you access Tanzu GemFire through the REST interface, objects are stored in Tanzu GemFire as 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. This interoperable format allows your Java applications to operate on the same data as your REST applications.

As an added benefit, because Tanzu GemFire's REST interface stores objects as PdxInstances, you do not need to write corresponding Java classes to translate JSON data (which you must do with other REST interface providers such as Oracle Coherence). For example, consider the use case where a non-Java REST client application (Python or Ruby) performs Tanzu GemFire region operations with JSON data that represents employee data. Since the object is stored in Tanzu GemFire as a PdxInstance that can be automatically mapped to JSON, the user does not need to write a corresponding Employee.java class and also does not need to worry about related issues such as keeping the Employee object in the CLASSPATH.

See Tanzu GemFire PDX Serialization for more information about PDX serialization.

Prerequisites and Limitations for Writing REST Applications

This topic describes prerequisites and limitations for writing VMware Tanzu GemFire REST applications.

Before development, you should understand the prerequisites and limitations of the Tanzu GemFire REST implementation.

Tanzu GemFire and REST-enabled applications accessing Tanzu GemFire are subject to the following rules and limitations:

- All domain objects, functions and function-arg classes must be properly configured and registered in the Tanzu GemFire deployment. Any functions that you wish to execute through the REST API must be available on the target member's CLASSPATH.
- The current implementation supports only the application/json MIME type. Other return types (XML, objects, and so on) are not supported. Plain text is supported as a return type for some error messages.
- Keys are strictly of type String. For example, the request `PUT http://localhost:8080/gemfire-api/v1/customers/123.456` will add an entry for key ("123.456") of type String.
- Some special formats of JSON documents are not supported in Tanzu GemFire REST. See Key Types and JSON Support for examples.
- To achieve interoperability between Tanzu GemFire Java clients (or Tanzu GemFire native clients) and REST clients, the following rules must be followed:
    - All Tanzu GemFire Java and native client classes operating on data also accessed by the REST interface must be PDX serialized, either via PDX autoserialization or by implementing `PdxSerializable`.
    - Tanzu GemFire Java clients and native clients can retrieve REST-enabled data either as a `PdxInstance` or as an actual object by using the `PdxInstance.getObject` method. If you use the latter method, you must first declare the object type (@type) in your POST or PUT request payload when creating the object in REST; and secondly, the Java client must have the actual domain class in its CLASSPATH.

- Objects returned by REST-invoked functions must be returned as PdxInstance objects or other data types that can be written to JSON. You cannot return Java objects.
- REST client applications do not support single hop access or notification features.
- Specifying subregions as endpoints is not supported.

Setup and Configuration

This topic explains setup and configuration of the VMware Tanzu GemFire Developer Rest API. The Tanzu GemFire Developer REST interface runs as an embedded HTTP or HTTPS service (Jetty server) within one or more Tanzu GemFire servers.

REST API Libraries

All Tanzu GemFire REST interface classes and required JAR files are distributed as a WAR file with the Tanzu GemFire product distribution. You can find the file in the following location:

install-dir/tools/Extensions/gemfire-web-api-*n.n.n*.war

where *install-dir* is the server installation directory and *n.n.n* is a version number.

Enabling the REST API

The REST API service for application development runs only on servers; you cannot run the service on a locator.

To enable the Developer REST API service on a given server, use the gfsh start server command with the --start-rest-api option. This starts an embedded Jetty server and deploys the Developer REST API WAR file on that server.

Enabling the REST API on Multiple Servers

You can configure multiple REST-enabled servers in a single cluster. Each server should have a separate host name and unique end point. To ensure that the server is reachable on a machine with multiple NIC addresses, use http-service-bind-address to bind an address to the REST API service (as well as the other embedded web services, such as Pulse).

You can configure the Developer REST API service to run over HTTPS by enabling SSL for the http component in gemfire.properties or gfsecurity.properties, or on server startup. See SSL for details on configuring SSL parameters. These SSL parameters apply to all HTTP services hosted on the configured server, which can include the following:

- Developer REST API service
- Management REST API service (for remote cluster management)
- Pulse monitoring tool.

> GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

Starting the REST API Service

To start a REST API service-enabled Tanzu GemFire deployment, configure PDX serialization for your cluster, then start the service on one or more server nodes.

Configure PDX for your cluster

You must configure PDX if either or both of the following conditions apply:

- Application peer member caches will access REST-accessible regions (resources) with Region.get(key).
- Your deployment has persistent regions that must be available as resources to the REST API.

To configure PDX in your cluster, perform the following steps:

1. Start a locator running the cluster configuration service (enabled by default). For example:

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

2. If your deployment has application peer member caches (for example, Java clients) that must also access REST-accessible Regions (resources), use the following gfsh command:

   ```
   gfsh>configure pdx --read-serialized=true
   ```

   > You do not need to configure --read-serialized=true if no application peer member caches are accessing the REST-accessible regions (resources) in your deployment.

3. If your deployment contains persistent regions that must be REST-accessible, use the following gfsh command:

   ```
   gfsh>configure pdx --disk-store
   ```

   This command sets pdx persistent equal to true and sets the disk-store-name to DEFAULT. If desired, specify an existing disk store name as the value for --disk-store.

4. If both of the above cases apply to your deployment, then configure PDX with the following single command:

   ```
   gfsh>configure pdx --read-serialized=true --disk-store
   ```

   After you have configured PDX for your caches, then proceed with starting your REST-enabled servers and other servers.

Start the REST API Service on One or More Servers

As described above, you can start the REST API service on a server by using gfsh start server --start-rest-api, or by setting the Tanzu GemFire property start-dev-rest-api to true. If you wish to start the service on multiple servers, use http-service-bind-address and http-service-port to identify the cache server and specific port that will host REST services. If you do not specify the http-service-port, the default port is 7070, which may collide with other locators and servers. If you do not specify http-service-bind-address, the HTTP service will bind to all local addresses by default.

> If your application will be running in a VM (as when running in the cloud, for example), it is good practice to specify http-service-bind-address and http-service-port so they will be publicly visible. The default values may not be visible outside the VM in which the application is running.

For example:

```
gfsh>start server --name=server1 --start-rest-api=true \
--http-service-port=8080 --http-service-bind-address=localhost
```

Any server that hosts data, even a server acting as a JMX manager, can start the Developer REST API service. For example, to start the service on a server that is also a JMX manager, you would run:

```
gfsh>start server --name=server1  --start-rest-api=true \
--http-service-port=8080 --http-service-bind-address=localhost \
--J=-Dgemfire.jmx-manager=true --J=-Dgemfire.jmx-manager-start=true
```

Note that when started as a JMX Manager, the server will also host the Pulse web application in the same HTTP service.

You may need to specify a class path to load any functions that need to be made available to your REST services. For example:

```
gfsh>start server --name=server1 --start-rest-api=true \
--http-service-port=8080 --http-service-bind-address=localhost \
--classpath=/myapps/testfunctions.jar
```

You can specify these properties either upon server startup or in the server's gemfire.properties configuration file. For example:

```
gfsh>start server --name=serverX --server-port=40405 --cache-xml-file=cache-config.xml \
--properties-file=gemfire.properties --classpath=/myapps/testfunctions.jar
```

where gemfire.properties contains:

```
http-service-port=8080
http-service-bind-address=localhost
start-dev-rest-api=true
```

Verify That The Service is Running

Verify that the Tanzu GemFire REST API service is up and running. To validate this, you can perform the following checks:

1. Test the list resources endpoint (this step assumes that you have regions defined on your cluster):

   ```
   curl -i http://localhost:8080/gemfire-api/v1
   ```

2. Examine the server logs for the following messages:

   ```
   [info 2017/06/13 13:48:14.090 PDT gfsec-server1 <main> tid=0x1] Initializing Spring FrameworkServlet 'gemfire-mgmt'
   [info 2017/06/13 13:48:14.091 PDT gfsec-server1 <main> tid=0x1] FrameworkServlet 'gemfire-mgmt': initialization start
   ed
   ```

3. Open a browser and enter the following URL to browse the Swagger-enabled REST APIs:

   ```
   http://<http-service-bind-address>:<http-service-port>/gemfire-api/docs/index.html
   ```

   where *http-service-bind-address* is the address and *http-service-port* is the port number that you specified when starting the Development REST API service on the server. For example, based on the server started in an earlier example, you would enter:

```
http://localhost:8080/gemfire-api/docs/index.html
```

If you did not specify these properties upon server startup or in gemfire.properties, then use the default of localhost and port 7070. See Using the Swagger UI to Browse REST APIs for more information.

Implementing Authentication

To turn on integrated security, start your servers and locators with the security-manager property set in your gemfire.properties file or on the gfsh command-line. The following example uses the sample implementation that is included in the Tanzu GemFire source, org.apache.geode.examples.security.ExampleSecurityManager.

This implementation requires a JSON security configuration file which defines the allowed users and their corresponding permissions. (See the javadocs for ExampleSecurityManager for details on how to compose the JSON file.) Place a copy of the JSON security configuration file in the execution directory of each security-enabled member, then specify --classpath=. in the start command for each of those members.

To start a server using a username and password that are defined in that server's security configuration, include the --user=username and --password=password options in the server's start command:

For example, suppose the JSON config file defines user "super-user" with password "1234567":

```
gfsh>start server --name=server1 --start-rest-api=true \
--http-service-port=8080 --http-service-bind-address=localhost \
--J=-Dgemfire.security-manager=org.apache.geode.examples.security.ExampleSecurityManager \
--classpath=. --user=super-user --password=1234567
```

To contact the server through the REST interface, you must provide the username and password. Various REST GUI interfaces provide different ways of accomplishing this. The curl command offers the --user (or -u) option for this purpose, where username and password are specified as a colon-separated pair:

```
curl -i --user super-user:1234567 http://localhost:8080/gemfire-api/v1
```

In a simple URL, such as in a browser address bar, the credentials can be given as a prefix to the host name in the form username:password@:

```
http://super-user:1234567@localhost:8080/gemfire-api/v1
```

Using the Swagger UI to Browse REST APIs

VMware Tanzu GemFire Developer REST APIs are integrated with the Swagger™ framework. This framework provides a browser-based test client that allows you to visualize and try Tanzu GemFire REST APIs.

Swagger application JARs are included in the GemFire REST application WAR. You do not need to install any additional libraries to use Swagger.

The following example demonstrates how to access the Swagger UI to browse the APIs.

1. Start a Tanzu GemFire Locator and a Developer REST API-enabled server as described in Setup and Configuration. Specify an http-service-port for the developer REST service, as the default port, 7070, is already taken by the locator. For example:

```
gfsh>start locator --name=locator1
Starting a GemFire Locator in /Users/admin/gemfire-1.2.0/locator1...
....
gfsh>start server --name=server1  --start-rest-api=true \
--http-service-bind-address=localhost --J=-Dgemfire.http-service-port=8080
```

2. To access Swagger, open a browser and enter the following URL. For example:

```
http://localhost:8080/gemfire-api/swagger-ui.html
```

The following Web page appears:

3. Using gfsh, create one or more regions on the REST API server. For example:

```
gfsh>create region --name=region1 --type=REPLICATE --key-constraint=java.lang.String
Member  | Status | Message
------- | ------ | ---------------------------------------
server1 | OK     | Region "/region1" created on "server1"

Cluster configuration for group 'cluster' is updated.
```

4. In Swagger, scroll down to region to view the list of all the available endpoints for accessing regions. (Click on region to expand the list of endpoints, if they are not already visible.)

5. In the list of region endpoints, click on the GET /v1 endpoint link. The page displays additional request and response information about the API.

6. Click the Try it out! button, then Execute. Any regions you added in step 3 are returned in the response body.

7. To add an entry to the region, expand the POST /v1/{region} endpoint.
8. Click the Try it out! button and fill in the parameters. This example uses:
   - region name: “region1”
   - key: “1”
   - Request body: a key/value pair specified as a JSON string, {"1" : "one"}.

9. Click Execute to see the response body and response code. The response indicates that the operation succeeded.

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/gemfire-api/v1/region1?key=1' \
  -H 'accept: application/json;charset=UTF-8' \
  -H 'Content-Type: application/json;charset=UTF-8' \
  -d '{
"1" : "one"
}'
```

Request URL

```
http://localhost:8080/gemfire-api/v1/region1?key=1
```

Server response

Code	Details
201	Response headers

```
cache-control: no-cache,no-store,max-age=0,must-revalidate
content-length: 0
content-security-policy: form-action 'self'
date: Wed,17 May 2023 21:14:35 GMT
expires: 0
location: http://localhost:8080/gemfire-api/v1/region1/1
pragma: no-cache
x-content-type-options: nosniff
x-frame-options: DENY
x-xss-protection: 1; mode=block
```

Responses

10. To view the new key and value, open the GET /v1/{region} endpoint. Click the Try it out! button and fill in the region name (“region1”). Then click Execute to see all keys and values for the region.

Parameters

Cancel

Name	Description
region * required string (path)	region1
limit string (query)	50
keys array[string] (query)	Add string item
ignoreMissingKey string (query)	ignoreMissingKey

Execute | Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/gemfire-api/v1/region1?limit=50' \
  -H 'accept: application/json;charset=UTF-8'
```

Request URL

```
http://localhost:8080/gemfire-api/v1/region1?limit=50
```

Server response

Code	Details
200	Response body

```
{
  "region1": [
    {
      "1": "one"
    }
  ]
}
```

Download

Response headers

```
cache-control: no-cache,no-store,max-age=0,must-revalidate
content-location: http://localhost:8080/gemfire-api/v1/region1/1
content-security-policy: form-action 'self'
content-type: application/json;charset=utf-8
date: Wed,17 May 2023 21:21:09 GMT
expires: 0
pragma: no-cache
transfer-encoding: chunked
x-content-type-options: nosniff
x-frame-options: DENY
x-xss-protection: 1; mode=block
```

You can use the Swagger interface to try out additional GemFire API endpoints and view sample responses.

For more information about Swagger, see the Swagger website and the OpenAPI specification.

Developing REST Applications

This topic provides guidelines on writing REST client applications for VMware Tanzu GemFire.

You can browse, query, update, and delete data stored in your Tanzu GemFire deployment. You can also manage and execute pre-deployed functions on Tanzu GemFire members.

- Working with Regions

  The Tanzu GemFire REST APIs provide basic CRUD (create, read, update and delete) operations for data entries stored in your regions.

- Working with Queries

  Tanzu GemFire supports the use of queries to extract data from its regions. Using REST APIs, you can create and execute either prepared or ad-hoc queries on Tanzu GemFire regions. You can also update and delete prepared queries.

- Working with Functions

  Tanzu GemFire REST APIs support the discovery and execution of predefined Tanzu GemFire functions on your cluster deployments.

Working with Regions

The Tanzu GemFire REST APIs provide basic CRUD (create, read, update and delete) operations for data entries stored in your regions.

Regions are the resources of the Tanzu GemFire REST API. Each region represents a resource or a collection of resources.

You cannot create or delete the regions themselves with the REST APIs, but you can work with the data stored within predefined Tanzu GemFire regions. Use the gfsh command utility to add, configure or delete regions in your Tanzu GemFire deployment. Any additions or modifications to regions made through gfsh are then accessible by the REST APIs.

Listing Available Regions

The main resource endpoint to the Tanzu GemFire API is GET /gemfire-api/v1. Use this endpoint to discover which regions are available in your cluster.

Example call:

```
curl -i http://localhost:7070/gemfire-api/v1
```

Example response:

```
Accept: application/json
Response Payload: application/json

200 OK
Server: Apache-Coyote/1.1
Location: http://localhost:7070/gemfire-api/v1
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 18 Jan 2014 20:05:47 GMT
{
  "regions": [
    {
      "name": "customers",
      "type": "REPLICATE",
      "key-constraint": "java.lang.String",
      "value-constraint": "org.apache.geode.pdx.PdxInstance"
    },
    {
      "name": "items",
      "type": "REPLICATE",
      "key-constraint": null,
      "value-constraint": null
    },
    {
      "name": "orders",
      "type": "PARTITION",
      "key-constraint": null,
      "value-constraint": null
    },
    {
      "name": "primitiveKVStore",
      "type": "PARTITION",
      "key-constraint": null,
      "value-constraint": null
    },
    {
      "name": "empty_region",
      "type": "EMPTY",
      "key-constraint": "java.lang.String",
      "value-constraint": "org.apache.geode.pdx.PdxInstance"
    }
  ]
}
```

Each region listed in the response includes the following region attributes:

- name. Name of the region.
- type. Type of region. For example, REPLICATE, PARTITION, or EMPTY. See Region Types for more information.
- key-constraint. If defined, the fully qualified class name of the key's type. Otherwise, null.
- value-constraint. If defined, the fully qualified class name of the value's type. Otherwise, null.

If no resources (regions) are available in the cluster, the call returns a 404 NOT FOUND error.

Reading Region Data

You can read data from a region by using any of the following REST-enabled mechanisms:

- GET /gemfire-api/v1/{region}?limit=ALL - Read all entries in a region

- GET /gemfire-api/v1/{region}?limit=*N* - Read a limited number of entries in a region
- GET /gemfire-api/v1/{region}/keys - List all keys in a region
- GET /gemfire-api/v1/{region}/{keys} - Read data for specific key or keys in a region

Reading Entries

To read entries in a region, use the following REST endpoint:

```
GET /gemfire-api/v1/{region}?[limit={<number>|ALL}]
```

For example:

```
http://localhost:7070/gemfire-api/v1/items
```

To read all entries in the region, specify instead:

```
http://localhost:7070/gemfire-api/v1/items?limit=ALL
```

To read 80 entries from the region, specify:

```
http://localhost:7070/gemfire-api/v1/items?limit=80
```

Setting the limit parameter is optional. If you do not specify a limit, the request will return 50 results by default. The returned order of the results is not guaranteed; however, the response values (JSON) are specified in the same order as the list of comma-separated keys returned in the “Content-location” header.

Reading Keys

To list all keys in a region, use the following endpoint:

```
/gemfire-api/v1/{region}/keys
```

For example:

```
http://localhost:7070/gemfire-api/v1/items/keys
```

You can use the returned key to perform additional operations on the keys such as read, update or delete their values.

Reading Entries By Key

To obtain data for a specific key or set of keys, use the following endpoints:

```
GET /gemfire-api/v1/{region}/{key1}
```

or

```
GET /gemfire-api/v1/{region}/{key1},{key2},...,{keyN}
```

where you specify keys in a comma-delimited list.

For example:

```
http://localhost:7070/gemfire-api/v1/items/1

http://localhost:7070/gemfire-api/v1/items/1,3,5
```

If one or more of the keys you provide in the list of keys is missing from the region, you will receive a 400 BAD STATUS error response.

If you are providing multiple keys, you can also use the ignoreMissingKey=true parameter to prevent 400 errors. Any non-existing keys will instead return a null response. For example:

```
http://localhost:7070/gemfire-api/v1/items/1,3,5?ignoreMissingKey=true
```

Adding or Modifying Region Data

To add data to a region, you have several options:

- Create new a brand new entry (both key and value) in a region
- Update or insert a value for a key (if the key does not exist, it will be created)
- Update (replace) data for a key if and only if the key already exists in region
- Compare existing value for a key before replacing value
- Update or insert multiple values in the region for a set of provided keys

Adding entries

To add a new key and value to a region, you can use the following endpoint:

```
POST /gemfire-api/v1/{region}?key=<key>
```

This endpoint only puts the entry into the region if the specified key does not already exist. If the key already exists, the request will fail with a 409 CONFLICT error.

> If you do not specify a key for this request, a String representation of a numerical key will be generated automatically for you.

Specify the value for the new entry in the request body. For example:

```
http://localhost:7070/gemfire-api/v1/orders?key=2
```

```
Request Payload: application/json
POST /gemfire-api/v1/orders?key=2

Accept: application/json
Content-Type: application/json
{
   "@type":  "org.apache.geode.web.rest.domain.Order",
    "purchaseOrderNo": 112,
    "customerId": 1012,
    "description":  "Purchase Order for myCompany",
    "orderDate":  "02/10/2014",
    "deliveryDate":  "02/20/2014",
    "contact":  "John Doe",
    "email":  "John.Doe@example.com",
    "phone":  "01-2048096",
    "totalPrice": 225,
    "items": [
      {
          "itemNo": 1,
          "description":  "Product2, PartA",
          "quantity": 10,
          "unitPrice": 5,
          "totalPrice": 50
      },
      {
          "itemNo": 2,
          "description":  "Product2, PartB",
          "quantity": 20,
          "unitPrice": 20,
          "totalPrice": 400
      }
   ]
}
```

Note that in the example above, @type is used to declare the value-constraint for the new entry. This declaration is required to provide interoperability between Java cache clients and REST clients.

Alternately, you can also use the following endpoint to create a new entry:

```
PUT /gemfire/v1/{region}/{key}
```

This endpoint will add the entry if the key does not exist. If the key already exists, the operation will update the entry.

```
http://localhost:7070/gemfire-api/v1/orders/2
```

```
Request Payload: application/json
PUT /gemfire-api/v1/orders/2
Request Payload: application/json
Content-Type: application/json
Accept: application/json
{
   "@type": "org.apache.geode.web.rest.domain.Order",
    "purchaseOrderNo": 1121,
    "customerId": 1012,
    "description":  "Order for XYZ Corp",
    "orderDate":  "02/10/2014",
    "deliveryDate":  "02/20/2014",
    "contact":  "Pie Doe",
    "email":  "pie.doe@example.com",
    "phone":  "01-2048096",
    "totalPrice": 225,
    "items": [
      {
          "itemNo": 1,
          "description": "Product-100",
          "quantity": 10,
          "unitPrice": 5,
          "totalPrice": 50
      }
   ]
}
```

Modifying existing entries

Tanzu GemFire provides three different options for this type of operation. To update a value for the key, you can use:

```
PUT /gemfire/v1/{region}/{key}
PUT /gemfire/v1{region}/{key}?op=REPLACE
PUT /gemfire/v1{region}/{key}?op=CAS
```

If you do not specify a parameter to the PUT operation, the request which will add or update the entry regardless of whether the key already exists in the region. See the example above.

```
http://localhost:7070/gemfire-api/v1/orders/2
```

If you specify the op=REPLACE parameter, the request which will explicitly perform a Cache replace operation and will verify that the key exists before replacing the value. If the key does not exist in the specified region, you will receive a 404 NOT FOUND error. This operation is idempotent, meaning multiple identical requests will have the same effect as the initial request.

```
http://localhost:7070/gemfire-api/v1/orders/2?op=REPLACE
```

If you specify the op=CAS parameter, the value will only be replaced with the @new value only if the specified @old value matches the current value of the key in the region. If the @old value does not match the current value, then a 409 CONFLICT error is thrown. If you receive a 409 CONFLICT error, you can call the GET /gemfire-api/v1/{region}/{key} endpoint to get an updated copy of the value. This operation is not idempotent and multiple identical requests will not have the same effect as the initial request. You can use this type of REST call to achieve a similar effect as optimistic locking.

```
http://localhost:7070/gemfire-api/v1/orders/2?op=REPLACE
```

```
Request Payload: application/json

PUT /gemfire-api/v1/orders/2?op=CAS

Accept: application/json
Content-Type: application/json
{
   "@old": {
       "@type":  "org.apache.geode.web.rest.domain.Order",
       "purchaseOrderNo": 1121,
       "customerId": 1012,
       "description":  "Order for  XYZ Corp",
       "orderDate":  "02/10/2014",
       "deliveryDate":  "02/20/2014",
       "contact":  "Jelly Bean",
       "email":  "jelly.bean@example.com",
       "phone":  "01-2048096",
       "items": [
         {
             "itemNo": 1,
             "description":  "Product-100",
             "quantity": 12,
             "unitPrice": 5,
             "totalPrice": 60
         }
       ],
       "totalPrice": 225
   },
    "@new ": {
       "@type":  "org.apache.geode.web.rest.domain.Order",
       "purchaseOrderNo": 1121,
       "customerId": 1013,
       "description":  "Order for  New Corp",
       "orderDate":  "02/10/2014",
       "deliveryDate":  "02/25/2014",
       "contact":  "Vanilla Bean",
       "email":  "vanillabean@example.com",
       "phone":  "01-2048096",
       "items": [
         {
             "itemNo": 12345,
             "description":  "part 123",
             "quantity": 12,
             "unitPrice": 29.99,
             "totalPrice": 149.95
         }
       ],
       "totalPrice": 149.95
   }
}
```

Adding or updating multiple values for a set of keys

To update multiple values for keys, use:

```
PUT /gemfire-api/v1/{region}/{key1},{key2},...,{keyN}
```

This REST call will update any keys that already exist and insert values for any keys that do not exist in the region.

Deleting Region Data

There are three options for deleting data in a region using REST APIs:

- Delete all the data in the region. Limited to replicated regions only; not available for partitioned regions.
- Delete the data associated with a particular key
- Delete the data associated with a set of keys

Deleting all data

To delete all data in the region, use the following endpoint:

```
DELETE /gemfire-api/v1/{region}
```

For example:

```
http://localhost:7070/gemfire-api/v1/items
```

Note that this does not delete the region itself, but instead all the entries in the region.

Deleting data based on key

Use:

```
DELETE /gemfire-api/v1/{region}/{key}
```

or

```
DELETE /gemfire-api/v1/{region}/{key}{key1},{key2},...{keyN}
```

If any of the supplied keys are not found in the region, the request will fail and return a 404 NOT FOUND ERROR.

Working with Queries

Tanzu GemFire supports the use of queries to extract data from its regions. Using REST APIs, you can create and execute either prepared or ad-hoc queries on Tanzu GemFire regions. You can also update and delete prepared queries.

Listing Queries

To find out which predefined and named queries are available in your deployment, use the following endpoint:

```
GET /gemfire-api/v1/queries
```

All queries that have been predefined and assigned IDs in Tanzu GemFire are listed.

Creating a New Query

To create a query, use the following endpoint:

```
POST /gemfire-api/v1/queries?id=<queryId>&q=<OQL-statement>
```

Here are some examples:

```
http://localhost:7070/gemfire-api/v1/queries?id=selectOffers&q="SELECT DISTINCT c FROM /customers c, /orders o WHERE o.totalprice < $1 AND c.customerId = o.customerId"
```

> The query must be provided as a URL parameter. You cannot specify OQL in the request body at this time.

You can specify query bind parameters ($1) in your predefined queries and then pass in values at runtime.

To update this query at a later time, use the PUT operation described below.

Executing a Prepared Query

To run a prepared query, use:

```
POST /gemfire-api/v1/queries/{queryId}
```

Specify the queryId in the URL. All query argument must be passed in the request body. For example:

```
http://localhost:7070/gemfire-api/v1/queries/selectOrders
```

```
Request Payload: OQL bind parameter values HTTP message body of media type application/json
POST /gemfire-api/v1/queries/selectOrders
Accept: application/json
Content-Type: application/json

[
  {
    "@type": "int",
    "@value": 2
  },
  {
    "@type": "double",
    "@value": 110.00
  }
]

Response Payload: application/json

200 OK
Content-Length: <#-of-bytes>
Content-Type: application/json

[
  {
    "description": "Purchase order for company - B",
    "totalPrice": 350,
    "purchaseOrderNo": 1112,
    "customerId": 102,
    "deliveryDate": "Thu Feb 20 00:00:00 IST 2014",
    "contact": "John Doe",
    "email": "jDoe@example.com",
    "phone": "01-2048096",
    "items": [
      {
        "description": "Product-AAAA",
        "quantity": 10,
        "itemNo": 1,
        "unitPrice": 20,
        "totalPrice": 200,
        "type-class": "org.apache.geode.web.rest.domain.Item"
      },
      {
        "description": "Product-BBB",
        "quantity": 15,
        "itemNo": 2,
        "unitPrice": 10,
        "totalPrice": 150,
        "type-class": "org.apache.geode.web.rest.domain.Item"
      }
    ],
    "orderDate": "Mon Feb 10 00:00:00 IST 2014",
    "type-class": "org.apache.geode.web.rest.domain.Order"
  },
  {...},
  {...}
}
```

Another example:

```
http://localhost:7070/gemfire-api/v1/queries/selectOrders
```

```
Request Payload: OQL bind parameter values HTTP message body of media type application/json
POST /gemfire-api/v1/queries/selectCustomer
Accept: application/json
Content-Type: application/json
{
   "args": [
      {
         "@type": "int",
         "@value": 101
      }
   ]
}

Response-Payload: application/json

200 Ok
Content-Length: 140
Content-Type: application/json
[
   {
      "firstName": "Jane",
      "lastName": "Doe",
      "customerId": 101,
      "type-class": "org.apache.geode.web.rest.domain.Customer"
   }
]
```

Modifying a Prepared Query

To modify an existing prepared query, use the following endpoint:

```
PUT /gemfire-api/v1/queries/{queryId}
```

Here are some examples:

```
http://localhost:7070/gemfire-api/v1/queries/selectOffers&q="SELECT DISTINCT c FROM /customers c, /orders o WHERE o.totalprice < $1 AND c.customerId = o.customerId"
```

You can specify query bind parameters ($1) in your predefined queries, and then pass in values at runtime.

A PUT operation will only succeed if the specified queryId already exists (for example, created with the POST operation above.) If the queryId does already exist, you will receive a 404 response - “Named query (selectKey456) does not exist!”

Deleting a Prepared Query

To delete an existing prepared query, use the following endpoint:

```
DELETE /gemfire-api/v1/queries/{queryId}
```

Executing an Ad-Hoc Query

To run an unnamed query, use the following endpoint:

```
GET /gemfire-api/v1/queries/adhoc?q=<OQL-statement>
```

Provide the OQL query string directly in the URL enclosed in single quotes.

For example:

```
http://localhost:7070/gemfire-api/v1/queries/adhoc?q="SELECT * FROM /customers"
```

Working with Functions

Tanzu GemFire REST APIs support the discovery and execution of predefined Tanzu GemFire functions on your cluster deployments.

Before you can access functions using REST APIs, you must have already defined and registered the functions in your Tanzu GemFire deployment. Additionally, any domain objects that are being accessed by the functions must be available on the class path of the server running the REST endpoint service.

You can do the following with functions:

- List all functions available in the Tanzu GemFire cluster.
- Execute a function, optionally specifying the region and members and/or member groups that are targeted by the function

Listing Functions

To list all functions that are currently registered and deployed in the Tanzu GemFire cluster, use the following endpoint:

```
GET /gemfire-api/v1/functions
```

The list of returned functions includes the functionId, which you can use to execute the function as described in the next section.

Executing Functions

To execute a function on a Tanzu GemFire cluster, use the following endpoint:

```
POST /gemfire-api/v1/functions/{functionId}?[&onRegion=regionname|&onMembers=member1,member2,...,memberN|&onGroups=group1,group2,...,groupN]
```

You have the option to target a specific region and members or member groups when executing your function. If you do not specify these parameters, the function will execute on all members that are hosting data in the entire cluster by default. Function arguments are passed in the request body.

For example:

```
http://localhost:7070/gemfire-api/v1/functions/AddFreeItemToOrders
```

```
Request Payload: application/json
POST /gemfire-api/v1/functions/AddFreeItemToOrders
Accept: application/json
Content-Type: application/json

[
  {
    "@type": "double",
    "@value": 210
  },
  {
    "@type": "org.apache.geode.web.rest.domain.Item",
    "itemNo": "599",
    "description": "Part X Free on Bumper Offer",
    "quantity": "2",
    "unitprice": "5",
    "totalprice": "10.00"
   }
]
```

In the above example, the Item domain object must be in the class paths of all members receiving the function. If the object is not defined, the function will fail with an Internal Server error. Look for ClassNotFoundExceptions in the stack trace.

Sample REST Applications

This topic provides examples that illustrate how multiple clients, both REST and native, can access the same VMware Tanzu GemFire region data.

> You must set PDX read-serialized to true when starting the cache server to achieve interoperability between different clients. See Setup and Configuration for instructions on starting up REST-enabled cache servers.

The following examples demonstrate the following:

1. A Java REST client creates a Person object on key 1. This client references the following supporting examples (also provided):
    1. Tanzu GemFire cache client
    2. REST client utility
    3. Date Time utility
    4. Person class
    5. Gender class
2. A Ruby REST client also gets data for key 1 and updates it.
3. A Python REST Client demonstrates the creation and modification of objects.

> An additional Python REST client reference application is available here: https://github.com/gemfire/py-gemfire-rest.

The following Java examples assume a project directory structure similar to the following:

#1. REST Java Client (RestClientApp.java)

```
package org.apache.geode.restclient;

 import org.springframework.http.HttpHeaders;
 import org.springframework.http.MediaType;
 import org.springframework.http.HttpMethod;
 import org.springframework.http.HttpEntity;
 import org.springframework.http.ResponseEntity;
 import org.springframework.web.client.HttpClientErrorException;
 import org.springframework.web.client.HttpServerErrorException;

 import org.apache.geode.util.RestClientUtils;

 import java.util.ArrayList;
 import java.util.List;

@SuppressWarnings( "unused")
 public class RestClientApp  {
   private static final String PEOPLE_REGION =  "/People";

   private static final String PERSON1_AS_JSON =  "{"
          +  "\"@type\ ": \"org.apache.geode.domain.Person\ "," +  "\"id\ ": 1,"
       +  " \"firstName\ ": \"Jane\ "," +  " \"middleName\ ": \"H\ ","
       +  " \"lastName\ ": \"Doe1\ "," +  " \"birthDate\ ": \"04/12/1983\ ","
       +  "\"gender\ ": \"MALE\ "" + "}";

   public static void main( final String... args)  throws Exception {
    doCreate(PEOPLE_REGION,  "1");
     System.out.println( "Programme has run successfully...!");
  }

   private static  HttpHeaders setAcceptAndContentTypeHeaders(){
    List<MediaType> acceptableMediaTypes =  new ArrayList<>();
    acceptableMediaTypes.add(MediaType.APPLICATION_JSON);

    HttpHeaders headers =  new HttpHeaders();
    headers.setAccept(acceptableMediaTypes);
    headers.setContentType(MediaType.APPLICATION_JSON);
     return headers;
  }

   private static void doCreate( final String regionNamePath,  final String key) {
    HttpHeaders headers =  setAcceptAndContentTypeHeaders();
    HttpEntity< String> entity =  new HttpEntity< String>(PERSON1_AS_JSON, headers);
     try {
      ResponseEntity< String> result = RestClientUtils.getRestTemplate().exchange(
            "http://localhost:8080/gemfire-api/v1/People?key=1" , HttpMethod.POST,
           entity,  String.class);

       System.out.println( "STATUS_CODE = " + result.getStatusCode().value());
       System.out.println( "HAS_BODY = " + result.hasBody());
       System.out.println( "LOCATION_HEADER = " + result.getHeaders().getLocation().toString());
    }  catch (HttpClientErrorException e) {
       System.out.println( "Http Client encountered error, msg:: " + e.getMessage());
    }  catch(HttpServerErrorException se) {
       System.out.println( "Server encountered error, msg::" + se.getMessage());
    }  catch (Exception e) {
       System.out.println( "Unexpected ERROR...!!");
    }
  }
}
```

#1a. Tanzu GemFire Cache Java Client (MyJavaClient.java)

```
package org.apache.geode.javaclient;

 import java.util.Calendar;
 import java.util.HashMap;
 import java.util.Map;

 import org.apache.geode.cache.Region;
 import org.apache.geode.cache.client.ClientCache;
 import org.apache.geode.cache.client.ClientCacheFactory;
 import org.apache.geode.cache.client.ClientRegionFactory;
 import org.apache.geode.cache.client.ClientRegionShortcut;
 import org.apache.geode.domain.Gender;
 import org.apache.geode.domain.Person;
 import org.apache.geode.pdx.PdxInstance;
 import org.apache.geode.util.DateTimeUtils;

 public class MyJavaClient {

   public static void main( String[] args) {
    ClientCacheFactory cf =  new ClientCacheFactory().addPoolServer( "localhost", 40405);
    ClientCache cache = cf.setPdxReadSerialized( true).create();
    ClientRegionFactory rf = cache.createClientRegionFactory(ClientRegionShortcut.PROXY);

    Region region = rf.create( "People");

     //Get data on key "1" , update it and put it again in cache
    Person actualObj =  null;
     Object obj = region.get( "1");
     if(obj  instanceof PdxInstance){
       System.out.println( "Obj is PdxInstance");
      PdxInstance pi = (PdxInstance)obj;
       Object obj2 = pi.getObject();
       if(obj2  instanceof Person){
        actualObj = (Person)obj2;
         System.out.println( "Received Person :" + actualObj.toString());
      } else {
         System.out.println( "Error: obj2 is expected to be of type Person");
      }
    } else {
       System.out.println( "Error: obj is expected to be of type PdxInstance");
    }

     //update the received object and put it in cache
 if(actualObj !=  null){
      actualObj.setFirstName( "Jane_updated");
      actualObj.setLastName( "Doe_updated");
      region.put( "1", actualObj);
    }

     //Add/putAll set of person objects
 final Person person2 =  new Person(102L,  "Sachin",  "Ramesh",  "Tendulkar", DateTimeUtils.createDate(1975, Calendar.DECE
MBER, 14), Gender.MALE);
     final Person person3 =  new Person(103L,  "Saurabh",  "Baburav",  "Ganguly", DateTimeUtils.createDate(1972, Calendar.AU
GUST, 29), Gender.MALE);
     final Person person4 =  new Person(104L,  "Rahul",  "subrymanyam",  "Dravid", DateTimeUtils.createDate(1979, Calendar.M
ARCH, 17), Gender.MALE);
     final Person person5 =  new Person(105L,  "Jhulan",  "Chidambaram",  "Goswami", DateTimeUtils.createDate(1983, Calenda
r.NOVEMBER, 25), Gender.FEMALE);
     final Person person6 =  new Person(101L,  "Rahul",  "Rajiv",  "Gndhi", DateTimeUtils.createDate(1970, Calendar.MAY, 14), G
ender.MALE);
     final Person person7 =  new Person(102L,  "Narendra",  "Damodar",  "Modi", DateTimeUtils.createDate(1945, Calendar.DEC
EMBER, 24), Gender.MALE);
     final Person person8 =  new Person(103L,  "Atal",  "Bihari",  "Vajpayee", DateTimeUtils.createDate(1920, Calendar.AUGUST,
```

```
9), Gender.MALE);
    final Person person9 =  new Person(104L,  "Soniya",  "Rajiv",  "Gandhi", DateTimeUtils.createDate(1929, Calendar.MARCH,
27), Gender.FEMALE);
    final Person person10 =  new Person(104L,  "Priyanka",  "Robert",  "Gandhi", DateTimeUtils.createDate(1973, Calendar.APRI
L, 15), Gender.FEMALE);
    final Person person11 =  new Person(104L,  "Murali",  "Manohar",  "Joshi", DateTimeUtils.createDate(1923, Calendar.APRIL,
25), Gender.MALE);
    final Person person12 =  new Person(104L,  "Lalkrishna",  "Parmhansh",  "Advani", DateTimeUtils.createDate(1910, Calenda
r.JANUARY, 01), Gender.MALE);
    final Person person13 =  new Person(104L,  "Shushma",  "kumari",  "Swaraj", DateTimeUtils.createDate(1943, Calendar.AUG
UST, 10), Gender.FEMALE);
    final Person person14 =  new Person(104L,  "Arun",  "raman",  "jetly", DateTimeUtils.createDate(1942, Calendar.OCTOBER,
27), Gender.MALE);
    final Person person15 =  new Person(104L,  "Amit",  "kumar",  "shah", DateTimeUtils.createDate(1958, Calendar.DECEMBE
R, 21), Gender.MALE);
    final Person person16 =  new Person(104L,  "Shila",  "kumari",  "Dixit", DateTimeUtils.createDate(1927, Calendar.FEBRUAR
Y, 15), Gender.FEMALE);

   Map< String,  Object> userMap =  new HashMap< String,  Object>();
   userMap.put( "2", person6);
   userMap.put( "3", person6);
   userMap.put( "4", person6);
   userMap.put( "5", person6);
   userMap.put( "6", person6);
   userMap.put( "7", person7);
   userMap.put( "8", person8);
   userMap.put( "9", person9);
   userMap.put( "10", person10);
   userMap.put( "11", person11);
   userMap.put( "12", person12);
   userMap.put( "13", person13);
   userMap.put( "14", person14);
   userMap.put( "15", person15);
   userMap.put( "16", person16);

    //putAll all person
   region.putAll(userMap);

    System.out.println( "successfully Put set of Person objects into the cache");
  }

}
```

#1b. REST Client Utilities (RestClientUtils.java)

```
package org.apache.geode.util;


import java.net.URI;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;
import org.springframework.http.converter.ByteArrayHttpMessageConverter;
import org.springframework.http.converter.HttpMessageConverter;
import org.springframework.http.converter.StringHttpMessageConverter;
import org.springframework.http.converter.json.Jackson2ObjectMapperFactoryBean;
import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
import org.springframework.web.client.RestTemplate;
import org.springframework.web.util.UriComponentsBuilder;

public class RestClientUtils {

  public static final String BASE_URL =  "http://192.0.2.0:8080" ;
  public static final String GEODE_REST_API_CONTEXT =  "/geode";
  public static final String GEODE_REST_API_VERSION =  "/v1";
  public static final URI GEODE_REST_API_WEB_SERVICE_URL = URI
      .create(BASE_URL + GEODE_REST_API_CONTEXT + GEODE_REST_API_VERSION);

  public static RestTemplate restTemplate;
  public static RestTemplate getRestTemplate() {
    if (restTemplate ==  null) {
     restTemplate =  new RestTemplate();
      final List<HttpMessageConverter<?>> messageConverters =  new ArrayList<HttpMessageConverter<?>>();

     messageConverters.add( new ByteArrayHttpMessageConverter());
     messageConverters.add( new StringHttpMessageConverter());
     messageConverters.add(createMappingJackson2HttpMessageConverter());

     restTemplate.setMessageConverters(messageConverters);
   }
    return restTemplate;
 }

  public static HttpMessageConverter< Object> createMappingJackson2HttpMessageConverter() {
    final Jackson2ObjectMapperFactoryBean objectMapperFactoryBean =  new Jackson2ObjectMapperFactoryBean();
   objectMapperFactoryBean.setFailOnEmptyBeans( true);
   objectMapperFactoryBean.setIndentOutput( true);
   objectMapperFactoryBean.setDateFormat( new SimpleDateFormat( "MM/dd/yyyy"));
   objectMapperFactoryBean
       .setFeaturesToDisable(com.fasterxml.jackson.databind.DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES);
   objectMapperFactoryBean
       .setFeaturesToEnable(
           com.fasterxml.jackson.core.JsonParser.Feature.ALLOW_COMMENTS,
           com.fasterxml.jackson.core.JsonParser.Feature.ALLOW_SINGLE_QUOTES,
           com.fasterxml.jackson.databind.DeserializationFeature.ACCEPT_EMPTY_STRING_AS_NULL_OBJECT);
   objectMapperFactoryBean.afterPropertiesSet();

    final MappingJackson2HttpMessageConverter httpMessageConverter =  new MappingJackson2HttpMessageConverter();
   httpMessageConverter.setObjectMapper(objectMapperFactoryBean.getObject());
    return httpMessageConverter;
 }

  public static URI toUri( final String... pathSegments) {
    return toUri(GEODE_REST_API_WEB_SERVICE_URL, pathSegments);
 }

  public static URI toUri( final URI baseUrl,  final String... pathSegments) {
    return UriComponentsBuilder.fromUri(baseUrl).pathSegment(pathSegments)
```

```
        .build().toUri();
  }
}
```

#1c. Date and Time Utilities (DateTimeUtils.java)

```
package org.apache.geode.util;


 import java.text.SimpleDateFormat;
 import java.util.Calendar;
 import java.util.Date;

/**
 * The DateTimeUtils class is a utility class  for working with dates and times.
 */
@SuppressWarnings( "unused")
 public abstract class DateTimeUtils {

   public static Calendar createCalendar( final int year,  final int month,  final int day) {
     final Calendar dateTime = Calendar.getInstance();
    dateTime.clear();
    dateTime.set(Calendar.YEAR, year);
    dateTime.set(Calendar.MONTH, month);
    dateTime.set(Calendar.DAY_OF_MONTH, day);
     return dateTime;
  }

   public static Date createDate( final int year,  final int month,  final int day) {
     return createCalendar(year, month, day).getTime();
  }

   public static String format( final Date dateTime,  final String formatPattern) {
     return (dateTime !=  null ?  new SimpleDateFormat(formatPattern).format(dateTime) :  null);
  }

}
```

#1d. Person Class (Person.java)

```
package org.apache.geode.domain;


 import java.util.Date;

 import org.apache.geode.internal.lang.ObjectUtils;
 import org.apache.geode.pdx.PdxReader;
 import org.apache.geode.pdx.PdxSerializable;
 import org.apache.geode.pdx.PdxWriter;

 import org.apache.geode.util.DateTimeUtils;

/**
 * The Person class is an abstraction modeling a person.
 */

 public class Person  implements PdxSerializable /*ResourceSupport  implements DomainObject< Long>*/  {

   private static final long serialVersionUID = 42108163264l;

   protected static final String DOB_FORMAT_PATTERN =  "MM/dd/yyyy";

   private Long id;

   private Date birthDate;

   private Gender gender;

   private String firstName;
   private String middleName;
   private String lastName;

   public Person() {
  }

   public Person( final Long id) {
     this.id = id;
  }

   public Person( final String firstName,  final String lastName) {
     this.firstName = firstName;
     this.lastName = lastName;
  }

   public Person( final Long id,  final String firstName,  final String middleName,  final String lastName, Date date, Gender gende
r) {
     this.id = id;
     this.firstName = firstName;
     this.middleName = middleName;
     this.lastName = lastName;
     this.birthDate = date;
     this.gender = gender;
  }

   public Long getId() {
     return id;
  }

   public void setId( final Long id) {
     this.id = id;
  }

   public String getFirstName() {
```

```
    return firstName;
  }

  public void setFirstName( final String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
    return lastName;
  }

  public void setLastName( final String lastName) {
    this.lastName = lastName;
  }

  public String getMiddleName() {
    return middleName;
  }

  public void setMiddleName( final String middleName) {
    this.middleName = middleName;
  }

  public Date getBirthDate() {
    return birthDate;
  }

  public void setBirthDate( final Date birthDate) {
    this.birthDate = birthDate;
  }

  public Gender getGender() {
    return gender;
  }

  public void setGender( final Gender gender) {
    this.gender = gender;
  }

  @Override
  public boolean equals( final Object obj) {
    if (obj ==  this) {
      return true;
    }

    if (!(obj  instanceof Person)) {
      return false;
    }

    final Person that = (Person) obj;

    return (ObjectUtils.equals( this.getId(), that.getId())
     || (ObjectUtils.equals( this.getBirthDate(), that.getBirthDate())
     && ObjectUtils.equals( this.getLastName(), that.getLastName())
     && ObjectUtils.equals( this.getFirstName(), that.getFirstName())));
  }

  @Override
  public int hashCode() {
    int hashValue = 17;
    hashValue = 37 * hashValue + ObjectUtils.hashCode(getId());
    hashValue = 37 * hashValue + ObjectUtils.hashCode(getBirthDate());
    hashValue = 37 * hashValue + ObjectUtils.hashCode(getLastName());
    hashValue = 37 * hashValue + ObjectUtils.hashCode(getFirstName());
```

```
    return hashValue;
  }

  @Override
   public String toString() {
    final StringBuilder buffer =  new StringBuilder( "{ type = ");
   buffer.append(getClass().getName());
   buffer.append( ", id = ").append(getId());
   buffer.append( ", firstName = ").append(getFirstName());
   buffer.append( ", middleName = ").append(getMiddleName());
   buffer.append( ", lastName = ").append(getLastName());
   buffer.append( ", birthDate = ").append(DateTimeUtils.format(getBirthDate(), DOB_FORMAT_PATTERN));
   buffer.append( ", gender = ").append(getGender());
   buffer.append( " }");
    return buffer.toString();
  }

  @Override
   public void fromData(PdxReader pr) {

   id = pr.readLong( "id");
   firstName = pr.readString( "firstName");
   middleName = pr.readString( "middleName");
   lastName = pr.readString( "lastName");
   birthDate = pr.readDate( "birthDate");
   gender = (Gender)pr.readObject( "gender");
  }

  @Override
   public void toData(PdxWriter pw) {
   pw.writeLong( "id", id);
   pw.writeString( "firstName", firstName);
   pw.writeString( "middleName", middleName);
   pw.writeString( "lastName", lastName);
   pw.writeDate( "birthDate", birthDate);
   pw.writeObject( "gender", gender);
  }

}
```

#1e. Gender Class (Gender.java)

```
package org.apache.geode.domain;

/**
 * The Gender  enum is a enumeration of genders (sexes).
 */

 public enum Gender {
  FEMALE,
  MALE
}
```

#2. Ruby REST Client (restClient.rb)

```
#!/usr/bin/ruby -w

puts "Hello, Ruby!";

# !/usr/bin/env ruby

require 'json'
require 'net/http'

class JsonSerializable

  def to_json
    hash = {}
    hash["@type"] = "org.apache.geode.web.rest.domain.Person"
    self.instance_variables.each do |var|
      if !var.to_s.end_with?("links")
        hash[var.to_s[1..-1]] = self.instance_variable_get var
      end
    end
    hash.to_json
  end

  def from_json! jsonString
    JSON.load(jsonString).each do |var, val|
      if !var.end_with?("type")
        self.instance_variable_set "@".concat(var), val
      end
    end
  end

end

class Person < JsonSerializable

  attr_accessor :id, :firstName, :middleName, :lastName, :birthDate, :gender

  def initialize(id = nil, firstName = nil, middleName = nil, lastName = nil )
    @id = id
    @firstName = firstName
    @middleName = middleName
    @lastName = lastName
    @birthDate = nil
    @gender = nil
  end

  def to_s
    s = "{ type = Person, id = #{@id}"
    s += ", firstName = #{@firstName}"
    s += ", middleName = #{@middleName}"
    s += ", lastName = #{@lastName}"
    s += ", birthDate = #{@birthDate}"
    s += ", gender = #{@gender}"
    s += "}"
  end

end

if __FILE__ == $0
  #p = Person.new(1, "Jon", "T", "Doe")
  #puts p
  #puts p.inspect
  #puts p.to_json
```

```
  uri = URI("http://localhost:8080/gemfire-api/v1/People/1");

  personJson = Net::HTTP::get(uri);

  # JSON from server
  puts "JSON read from Server for Person with ID 1...\n #{personJson}"

  p = Person.new
  p.from_json! personJson

  # print the Person to standard out
  puts "Person is...\n #{p}"

  p.id = 1
  p.firstName = "Jack"
  p.lastName = "Handy"
  p.gender = "MALE"

  # prints modified Person to standard out
  puts "Person modified is...\n #{p}"

  puts "JSON sent to Server for Person with ID 1...\n #{p.to_json}"

  Net::HTTP.start(uri.hostname, uri.port) do |http|
    putRequest = Net::HTTP::Put.new uri.path, { "Content-Type" => "application/json" }
    putRequest.body = p.to_json
    http.request(putRequest)
  end

end
```

Output from running the Ruby client:

```
prompt# ruby restClient.rb
Hello, Ruby!
JSON read from Server for Person with ID 1...
 {
  "@type" : "org.gopivotal.app.domain.Person",
  "id" : 1,
  "firstName" : "Jane_updated",
  "middleName" : "H",
  "lastName" : "Doe_updated",
  "gender" : "MALE",
  "birthDate" : "04/12/1983"
}
Person is...
 { type = Person, id = 1, firstName = Jane_updated, middleName = H, lastName = Doe_updated, birthDate = 04/12/1983, gender
= MALE}
Person modified is...
 { type = Person, id = 1, firstName = Jack, middleName = H, lastName = Handy, birthDate = 04/12/1983, gender = MALE}
JSON sent to Server for Person with ID 1...
 {"@type":"org.apache.geode.web.rest.domain.Person","id":1,"firstName":"Jack","middleName":"H","lastName":"Handy","birthDat
e":"04/12/1983","gender":"MALE"}
```

#3. Python REST Client (restClient.py)

This example uses Python 3 and shows the creation and modification of objects. It uses one external library called requests, which is nearly ubiquitous and avoids having to use HTTP code.

```
#!/usr/bin/env python3

# This is simple, repetitive and assumes you have created a region called
# "demoRegion".

import sys
import json
import uuid
import requests

REGION = "demoRegion"
BASE_URI = "http://localhost:8080/gemfire-api/v1"

headers = {'content-type': 'application/json'}

person = {'type': 'Person',
          'firstName': 'John',
          'middleName': 'Q',
          'lastName': 'Public',
          'birthDate': '1 Jan 1900'}


def resource_uri(res=None, region=REGION):
    if res:
        return "%s/%s/%s" % (BASE_URI, region, res)
    return "%s/%s" % (BASE_URI, region)


print("[*] First, we'll empty out our demo region - DELETE %s" %
      requests.delete(resource_uri()))

r = requests.delete(resource_uri())
r.raise_for_status()

print("[*] Now, we'll create 5 demo objects")

keys = []

for i in range(1, 6):
    key = uuid.uuid1()

    keys.append(key)
    person['uuid'] = str(key)

    print("\t Creating object with key: POST %s" % key)
    r = requests.post(resource_uri(), data=json.dumps(person),
                      params={'key': key},
                      headers=headers)
    r.raise_for_status()

print("[*] List our keys - GET %s" % resource_uri("keys"))

r = requests.get(resource_uri("keys"))
print(r.text)

print("[*] Here's all our data - GET %s" % resource_uri())

r = requests.get(resource_uri())
print(r.text)

print("[*] Now each key one by one")

for key in keys:
```

```
    print("Fetching key - GET %s" % resource_uri(res=key))
    r = requests.get(resource_uri(res=key))
    print(r.text)

print("[*] Now grab one, change the first name to 'Jane' and save it")

print("  GET - %s" % resource_uri(res=keys[0]))
r = requests.get(resource_uri(res=keys[0]))
p = json.loads(r.text)
p['firstName'] = 'Jane'
print("  PUT - %s" % resource_uri(res=keys[0]))
r = requests.put(resource_uri(res=keys[0]), data=json.dumps(p),
                 headers=headers)

print("  GET - %s" % resource_uri(res=keys[0]))
r = requests.get(resource_uri(res=keys[0]))
print(r.text)
```

Troubleshooting and FAQ

This topic provides troubleshooting guidance and frequently asked questions about VMware Tanzu GemFire Developer REST APIs.

Checking if the REST API Service is Up and Running

Use the ping endpoint to verify whether the REST API server is available.

Use the /gemfire-api/v1/ping endpoint to check REST API server availability:

For example:

```
curl -i http://localhost:7070/gemfire-api/v1/ping
```

Example success response:

```
200 OK
```

If the server is not available, your client will receive an HTTP error code and message.

Key Types and JSON Support

When defining regions (your REST resources), you must only use scalar values for keys and also set value constraints in order to avoid producing JSON that cannot be parsed by Tanzu GemFire.

If Tanzu GemFire regions are not defined with scalar values as keys and value constraints, then you may receive the following error message (even though the JSON is technically valid) in your REST client applications:

```
Json doc specified in request body is malformed..!!'
```

For example, the following JSON documents are not supported by Tanzu GemFire:

Unsupported JSON Example 1

```
[
    1,
    [],
    [
        4,
        "hello",
        {}
    ],
    {
        "array": []
    }
]
```

Unsupported JSON Example 2

```
[[[[[[[[[[[[[[[[[[[["Not too deep"]]]]]]]]]]]]]]]]]]]]
```

Unsupported JSON Example 3

```
[1,2,3,"hello"]
```

Unsupported JSON Example 4

```
[
    "JSON Test Pattern pass1",
    {
        "object with 1 member": [
            "array with 1 element"
        ]
    },
    {},
    [],
    -42,
    true,
    false,
    null,
    {
        "integer": 1234567890,
        "real": -9876.54321,
        "e": 1.23456789e-13,
        "E": 1.23456789e+34,
        "": 2.3456789012e+76,
        "zero": 0,
        "one": 1,
        "space": " ",
        "quote": "\"",
        "backslash": "\",
        "controls": "\b\f\n\r\t",
        "slash": "/ & /",
        "alpha": "abcdefghijklmnopqrstuvwyz",
        "ALPHA": "ABCDEFGHIJKLMNOPQRSTUVWYZ",
        "digit": "0123456789",
        "0123456789": "digit",
        "special": "`1~!@#$%^&*()_+-={':[,]}|;.</>?",
        "true": true,
        "false": false,
        "null": null,
        "array": [],
        "object": {},
        "address": "50 St. James Street",
        "url": "http://www.JSON.org/",
        "comment": "// /* <!-- --",
        "# -- --> */": " ",
        " s p a c e d ": [
            1,
            2,
            3,
            4,
            5,
            6,
            7
        ],
        "compact": [
            1,
            2,
            3,
            4,
            5,
            6,
            7
        ],
        "jsontext": "{\"object with 1 member\":[\"array with 1 element\"]}",
        "quotes": "&#34; \" %22 0x22 034 &#x22;",
        "/\\"\b\f\n\r\t`1~!@#$%^&*()_+-=[]{}|;:',./<>?": "A key can be any string"
    },
    0.5,
```

```
  98.6,
  99.44,
  1066,
  10,
  1,
  0.1,
  1,
  2,
  2,
  "rosebud"
]
```

Tanzu GemFire REST API Reference

This topic summarizes all available VMware Tanzu GemFire REST API resources and endpoints.

> This documentation covers the v1 release of Tanzu GemFire REST APIs for developing applications.

- Region Endpoints

  A Tanzu GemFire region is how Tanzu GemFire logically groups data within its cache. Regions stores data as entries, which are key-value pairs. Using the REST APIs you can read, add (or update), and delete region data.

- Query Endpoints

  Tanzu GemFire uses a query syntax based on OQL (Object Query Language) to query region data. Since Tanzu GemFire regions are key-value stores, values can range from simple byte arrays to complex nested objects.

- Function Endpoints

  Tanzu GemFire functions allows you to write and execute server-side transactions and data operations. These may include anything ranging from initializing components or third-party services or aggregating data.

- Administrative Endpoints

  Administrative endpoints provide management and monitoring functionality for the REST API interface.

Region Endpoints

This topic provides links to information about VMware Tanzu GemFire regions. Tanzu GemFire logically groups data within its cache in Tanzu GemFire Regions. Regions stores data as entries, which are key-value pairs. Using the REST APIs you can read, add, update, and delete region data.

For more in information about working with regions, see Data Regions.

- GET /gemfire-api/v1

  List all available resources (regions) in the Tanzu GemFire cluster.

- GET /gemfire-api/v1/{region}

  Read data for the region. The optional limit URL query parameter specifies the number of values from the Region that will be returned. The default limit is 50. If the user specifies a limit of “ALL”, then all entry values for the region will be returned.

- GET /gemfire-api/v1/{region}/keys

  List all keys for the specified region.

- GET /gemfire-api/v1/{region}/{key}

  Read data for a specific key in the region.

- GET /gemfire-api/v1/{region}/{key1},{key2},…,{keyN}

  Read data for multiple keys in the region.

- HEAD /gemfire-api/v1/{region}

  An HTTP HEAD request that returns region's size (number of entries) within the HEADERS, which is a response without the content-body. Region size is specified in the pre-defined header named "Resource-Count".

- POST /gemfire-api/v1/{region}?key=<key>

  Create (put-if-absent) data in region.

- PUT /gemfire-api/v1/{region}/{key}

  Update or insert (put) data for key in region.

- PUT /gemfire-api/v1/{region}/{key1},{key2},…{keyN}

  Update or insert (put) data for multiple keys in the region.

- PUT /gemfire-api/v1/{region}/{key}?op=REPLACE

  Update (replace) data with keys if and only if the keys exists in region. The Keys must be present in the Region for the update to occur.

- PUT /gemfire-api/v1/{region}/{key}?op=CAS

  Update (compare-and-set) value having key with a new value if and only if the "@old" value sent matches the current value having key in region.

- DELETE /gemfire-api/v1/{region}

  Delete all entries in the region.

- DELETE /gemfire-api/v1/{region}/{key}

  Delete entry for specified key in the region.

- DELETE /gemfire-api/v1/{region}/{key1},{key2},…{keyN}

  Delete entries for multiple keys in the region.

GET /gemfire-api/v1

This topic describes the GET command in VMware Tanzu GemFire to list all available resources (regions) in the Tanzu GemFire cluster.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1
```

Parameters

None.

Example Request

```
GET /gemfire/v1/
Accept: application/json
```

Example Success Response

```
Response Payload: application/json

200 OK
Server: Apache-Coyote/1.1
Location: http://localhost:8080/gemfire-api/v1
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 18 Jan 2020 20:05:47 GMT
 {
   "regions": [
      {
         "name": "customers",
         "type": "REPLICATE",
         "key-constraint": "java.lang.String",
         "value-constraint": "org.apache.geode.pdx.PdxInstance"
      },
      {
         "name": "items",
         "type": "REPLICATE",
         "key-constraint": null,
         "value-constraint": null
      },
      {
         "name": "orders",
         "type": "PARTITION",
         "key-constraint": null,
         "value-constraint": null
      },
      {
         "name": "primitiveKVStore",
         "type": "PARTITION",
         "key-constraint": null,
         "value-constraint": null
      },
      {
         "name": "empty_region",
         "type": "EMPTY",
         "key-constraint": "java.lang.String",
         "value-constraint": "org.apache.geode.pdx.PdxInstance"
      }
   ]
}
```

Error Codes

Status Code	Description
404 NOT FOUND	No regions were found at the provided endpoint.
500 INTERNAL SERVER ERROR	Encountered error at server.

GET /gemfire-api/v1/{region}

This topic describes the GET command in VMware Tanzu GemFire to read data for the region. The optional limit URL query parameter specifies the number of values from the Region that will be returned. The default limit is 50. If the user specifies a limit of ALL, then all entry values for the region will be returned.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/{region}?[limit={<number>|ALL}]
```

Parameters

Parameter	Description	Example Values
limit	Optional. Specify a limit to the number of region entries to return. If the limit parameter is not specified, the default is to return 50 results. Default: 50	ALL 80

Example Request

```
GET /gemfire-api/v1/orders/
Accept: application/json
```

Example Success Response

```
Response Payload: application/json

200 OK
Server: Apache-Coyote/1.1
Content-Location: http://localhost:8080/gemfire-api/v1/orders/3,1
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 18 Jan 2020 21:03:08 GMT

{
  "orders" : [ {
    "purchaseOrderNo" : 1112,
    "customerId" : 102,
    "description" :  "Purchase order for  company - B",
    "orderDate" :  "02/10/2014",
    "deliveryDate" :  "02/20/2014",
    "contact" :  "John Doe",
    "email" :  "John.Doe@example.com",
    "phone" :  "01-2048096",
    "items" : [ {
      "itemNo" : 1,
      "description" :  "Product-AAAA",
      "quantity" : 10,
      "unitPrice" : 20.0,
      "totalPrice" : 200.0
    }, {
      "itemNo" : 2,
      "description" :  "Product-BBB",
      "quantity" : 15,
      "unitPrice" : 10.0,
      "totalPrice" : 150.0
    } ],
    "totalPrice" : 350.0
  }, {
    "purchaseOrderNo" : 111,
    "customerId" : 101,
    "description" :  "Purchase order for  company - A",
    "orderDate" :  "01/10/2014",
    "deliveryDate" :  "01/20/2014",
    "contact" :  "Jane Doe",
    "email" :  "Jane.Doe@example.com",
    "phone" :  "020-2048096",
    "items" : [ {
      "itemNo" : 1,
      "description" :  "Product-1",
      "quantity" : 5,
      "unitPrice" : 10.0,
      "totalPrice" : 50.0
    }, {
      "itemNo" : 1,
      "description" :  "Product-2",
      "quantity" : 10,
      "unitPrice" : 15.5,
      "totalPrice" : 155.0
    } ],
    "totalPrice" : 205.0
  } ]
}
```

Error Codes

Status Code	Description
400 BAD REQUEST	Limit parameter X is not valid! The specified limit value must be ALL or an integer.
404 NOT FOUND	Returned if region does not exist.
500 INTERNAL ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception.

GET /gemfire-api/v1/{region}/keys

This topic describes the GET command in VMware Tanzu GemFire to list all keys for the specified region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/{region}/{keys}
```

Parameters

None.

Example Request

```
GET /gemfire-api/v1/orders/keys
```

Example Success Response

```
Response Payload: application/json

200 OK
Server: Apache-Coyote/1.1
Location: http://localhost:8080/gemfire-api/v1/orders/keys
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 18 Jan 2020 21:20:05 GMT
{   "keys": [
      "1",
      "2",
      "3"
   ]
}
```

Error Codes

Status Codes	Description
404 NOT FOUND	Specified region does not exist.
405 METHOD NOT ALLOWED	Returned if any HTTP request method other than GET (for example, POST, PUT, DELETE, etc.) is used.
500 INTERNAL SERVER ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception.

GET /gemfire-api/v1/{region}/{key}

This topic describes the GET command in VMware Tanzu GemFire to read data for a specific key in the region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/{region}/{key}
```

Parameters

None.

Example Request

```
GET /gemfire-api/v1/orders/1

Request Payload: null
Accept: application/json
```

Example Responses

```
Response Payload: application/json

200 OK
Server: Apache-Coyote/1.1
Content-Location: http: //localhost:8080/gemfire-api/v1/orders/1
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 18 Jan 2020 21:27:59 GMT

{
  "purchaseOrderNo" : 111,
  "customerId" : 101,
  "description" :  "Purchase order for  company - A",
  "orderDate" :  "01/10/2020",
  "deliveryDate" :  "01/20/2020",
  "contact" :  "Jane Doe",
  "email" :  "Jane.Doe@example.com",
  "phone" :  "020-2048096",
  "items" : [ {
    "itemNo" : 1,
    "description" :  "Product-1",
    "quantity" : 5,
    "unitPrice" : 10.0,
    "totalPrice" : 50.0
  }, {
    "itemNo" : 1,
    "description" :  "Product-2",
    "quantity" : 10,
    "unitPrice" : 15.5,
    "totalPrice" : 155.0
  } ],
  "totalPrice" : 205.0
}
```

Error Codes

Status Code	Description
400 BAD REQUEST	Returned if the supplied key is not found in the region.
404 NOT FOUND	Returned if the region or specified key is not found.
500 INTERNAL SERVER ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception.

GET /gemfire-api/v1/{region}/{key1},{key2},…,{keyN}

This topic describes the GET command in VMware Tanzu GemFire to read data for multiple keys in the region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/queries[?ignoreMissingKey=true]
```

Parameters

Parameter	Description	Example Values
ignoreMissingKey	Optional. Boolean to indicate whether to return non-existing keys as null responses. Use this to prevent 400 errors.	true false

Example Requests

```
GET /gemfire-api/v1/orders/1,2,13,4,5?ignoreMissingKey=true
Request Payload: null
Accept: application/json


GET /gemfire-api/v1/orders/1,3
Request Payload: null
Accept: application/json
```

Example Success Responses

Response Payload: application/json

200 OK
Server: Apache-Coyote/1.1
Content-Location: http://localhost:8080/gemfire-api/v1/orders/13,2,1,5,4
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 18 Jan 2020 21:39:04 GMT


```
{
  "orders": [
    null,
    {
      "@type": "org.apache.geode.rest.internal.web.controllers.Order",
      "purchaseOrderNo": 1112,
      "customerId": 102,
      "description": "Purchase order for company - B",
      "orderDate": "02/10/2020",
      "deliveryDate": "02/20/2020",
      "contact": "John Doe",
      "email": "john.doe@example.com",
      "phone": "01-2048096",
      "totalPrice": 350,
      "items": [
        {
          "itemNo": 1,
          "description": "Product-AAAA",
          "quantity": 10,
          "unitPrice": 20,
          "totalPrice": 200
        },
        {
          "itemNo": 2,
          "description": "Product-BBB",
          "quantity": 15,
          "unitPrice": 10,
          "totalPrice": 150
        }
      ]
    },
    {
      "@type": "org.apache.geode.rest.internal.web.controllers.Order",
      "purchaseOrderNo": 11101,
      "customerId": 101,
      "description": "Purchase order for company - A",
      "orderDate": "01/10/2020",
      "deliveryDate": "01/20/2020",
      "contact": "Jane Doe",
      "email": "jane.doe@example.com",
      "phone": "020-2048096",
      "totalPrice": 205,
      "items": [
        {
          "itemNo": 1,
          "description": "Product-1",
          "quantity": 5,
          "unitPrice": 10,
          "totalPrice": 50
        },
        {
          "itemNo": 3,
          "description": "Product-3",
```

```
                "quantity": 10,
                "unitPrice": 100,
                "totalPrice": 1000
            },
            {
                "itemNo": 1,
                "description": "Product-2",
                "quantity": 10,
                "unitPrice": 15.5,
                "totalPrice": 155
            }
        ]
    },
    null,
    null
  ]
}
```


200 OK
Server: Apache-Coyote/1.1
Content-Location: http://localhost:8080/gemfire-api/v1/orders/3,1
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 18 Jan 2020 21:39:04 GMT


```
{
  "orders" : [ {
    "purchaseOrderNo" : 1112,
    "customerId" : 102,
    "description" : "Purchase order for company - B",
    "orderDate" : "02/10/2020",
    "deliveryDate" : "02/20/2020",
    "contact" : "John Doe",
    "email" : "john.doe@example.com",
    "phone" : "01-2048096",
    "items" : [ {
      "itemNo" : 1,
      "description" : "Product-AAAA",
      "quantity" : 10,
      "unitPrice" : 20.0,
      "totalPrice" : 200.0
    }, {
      "itemNo" : 2,
      "description" : "Product-BBB",
      "quantity" : 15,
      "unitPrice" : 10.0,
      "totalPrice" : 150.0
    } ],
    "totalPrice" : 350.0
  }, {
    "purchaseOrderNo" : 111,
    "customerId" : 101,
    "description" : "Purchase order for company - A",
    "orderDate" : "01/10/2020",
    "deliveryDate" : "01/20/2020",
    "contact" : "Jane Doe",
    "email" : "jane.doe@example.com",
    "phone" : "020-2048096",
    "items" : [ {
      "itemNo" : 1,
      "description" : "Product-1",
      "quantity" : 5,
      "unitPrice" : 10.0,
```

```
    "totalPrice" : 50.0
  }, {
    "itemNo" : 1,
    "description" : "Product-2",
    "quantity" : 10,
    "unitPrice" : 15.5,
    "totalPrice" : 155.0
  } ],
  "totalPrice" : 205.0
 } ]
}
```

Error Codes

Status Code	Description
400 BAD REQUEST	Returned if one or more of the supplied keys is not found in the region and ignoreMissingKey=false (default if parameter is not specified); returned if the specified value for the ignoreMissingKey parameter is a value other than ‘true’ or ‘false’.
404 NOT FOUND	The specified region does not exist.

Example Error Response

```
GET /gemfire-api/v1/orders/1,2,13,4,5
Request Payload: null
Accept: application/json
Response Payload: application/json

400 BAD_REQUEST
Server: Apache-Coyote/1.1
Content-Type: application/json
Content-Length: 51
Date: Wed, 21 May 2020 11:24:00 GMT
Connection: close

"Requested keys [13,4,5] not exist in region [orders]"
```

Implementation Notes

The response body sets the region name as key to the array of values. For example “orders” : <VALUE_ARRAY>.

HEAD /gemfire-api/v1/{region}

This topic describes an HTTP HEAD request in VMware Tanzu GemFire that returns region’s size (number of entries) within the HEADERS, which is a response without the content-body. Region size is specified in the pre-defined header named “Resource-Count”.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/{region}
```

Parameters

None.

Example Request

```
Request Payload: null
HEAD /gemfire-api/v1/customers
Response Payload: null
```

Example Success Response

```
200 OK

Content-Length: <#-of-bytes>
Content-Type: application/json; charset=utf-8
...

Resource-Count: 8192
```

Error Codes

Status Code	Description
400 Bad Request	Returned if Tanzu GemFire throws an error while executing the request.
404 Resource Not Found	Region does not exist.
500 Internal Server Error	Tanzu GemFire has thrown an error or exception.

POST /gemfire-api/v1/{region}?key=

This topic describes the POST command in VMware Tanzu GemFire to create (put-if-absent) data in a region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/{region}?key=<key>
```

Parameters

Parameter	Description	Example Values
key	Optional. If you do not specify a key in the URL, a key will automatically be generated for you.	123 myOrder

Example Request

```
Request Payload: application/json
POST /gemfire-api/v1/orders?key=2

Accept: application/json
Content-Type: application/json
{
   "@type":  "org.apache.geode.web.rest.domain.Order",
    "purchaseOrderNo": 112,
    "customerId": 1012,
    "description":  "Purchase Order for myCompany",
    "orderDate":  "02/10/2020",
    "deliveryDate":  "02/20/2020",
    "contact":  "John Doe",
    "email":  "John.Doe@example.com",
    "phone":  "01-2048096",
    "totalPrice": 225,
    "items": [
      {
          "itemNo": 1,
          "description":  "Product2, PartA",
          "quantity": 10,
          "unitPrice": 5,
          "totalPrice": 50
      },
      {
          "itemNo": 2,
          "description":  "Product2, PartB",
          "quantity": 20,
          "unitPrice": 20,
          "totalPrice": 400
      }
   ]
}
```

Example Success Response

```
Response Payload: null
201 CREATED
Location: http://localhost:8080/gemfire-api/v1/orders/2
```

Error Codes

Status Code	Description
400 BAD REQUEST	Returned if JSON content is malformed.
404 NOT FOUND	Returned if the specified region does not exist.
409 CONFLICT	Returned if the provided key already exists in the region.
500 INTERNAL SERVER ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception.

Example Error Response

```
409 Conflict
Location:http://localhost:8080/gemfire-api/v1/orders/2
Content-Type: application/json

{
    "purchaseOrderNo": 112,
    "customerId": 1012,
    "description":  "Purchase Order for myCompany",
    "orderDate":  "02/10/2020",
    "deliveryDate":  "02/20/2020",
    "contact":  "John Doe",
    "email":  "John.Doe@example.com",
    "phone":  "01-2048096",
    "totalPrice": 225,
    "items": [
      {
          "itemNo": 1,
          "description":  "Product2, PartA",
          "quantity": 10,
          "unitPrice": 5,
          "totalPrice": 50
      },
      {
          "itemNo": 2,
          "description":  "Product2, PartB",
          "quantity": 20,
          "unitPrice": 20,
          "totalPrice": 400
      }
   ]
}
```

PUT /gemfire-api/v1/{region}/{key}

This topic describes the PUT command in VMware Tanzu GemFire to update or insert data for a key in a region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/{region}/{key}
```

Parameters

See PUT /gemfire-api/v1/{region}/{key}?op=REPLACE and PUT /gemfire-api/v1/{region}/{key}?op=CAS.

Example Request

```
PUT /gemfire-api/v1/orders/2
Request Payload: application/json
Content-Type: application/json
Accept: application/json
{
   "@type": "org.apache.geode.web.rest.domain.Order",
    "purchaseOrderNo": 1121,
    "customerId": 1012,
    "description":  "Order for XYZ Corp",
    "orderDate":  "02/10/2020",
    "deliveryDate":  "02/20/2020",
    "contact":  "Pie Doe",
    "email":  "pie.doe@example.com",
    "phone":  "01-2048096",
    "totalPrice": 225,
    "items": [
      {
          "itemNo": 1,
          "description": "Product-100",
          "quantity": 10,
          "unitPrice": 5,
          "totalPrice": 50
      }
   ]
}
```

Example Success Response

```
Response Payload:  null

200 OK
```

Error Codes

Status Code	Description
400 BAD REQUEST	Returned if supplied key is an invalid format.
404 NOT FOUND	Returned if the region is not found.
500 INTERNAL SERVER ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception.

Implementation Notes

This operation is idempotent, meaning multiple identical requests should have the same effect as the initial request.

PUT /gemfire-api/v1/{region}/{key1},{key2},…{keyN}

This topic describes the PUT command in VMware Tanzu GemFire to update or insert data for multiple keys in a region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/{region}/{key},{key2},...{keyN}
```

Parameters

See PUT /gemfire-api/v1/{region}/{key}?op=REPLACE and PUT /gemfire-api/v1/{region}/{key}?op=CAS.

Example Request

```
Request Payload: application/json

PUT /gemfire-api/v1/orders/4,5

Accept: application/json
Content-Type: application/json
[
  {
    "@type":  "org.apache.geode.web.rest.domain.Order",
    "purchaseOrderNo": 555,
    "customerId": 5,
    "description":  "Order for 23 Corp",
    "orderDate":  "01/15/2020",
    "deliveryDate":  "02/20/2020",
    "contact":  "Jane Doe",
    "email":  "jane.doe@example.com",
    "phone":  "01-2048096",
    "items": [
      {
        "itemNo": 321,
        "description":  "part 21",
        "quantity": 2,
        "unitPrice": 49.95,
        "totalPrice": 99.9
      }
    ],
    "totalPrice": 99.9
  },
  {
    "@type":  "org.apache.geode.web.rest.domain.Order",
    "purchaseOrderNo": 777,
    "customerId": 11,
    "description":  "Order for  MNP Corp",
    "orderDate":  "02/10/2020",
    "deliveryDate":  "02/20/2020",
    "contact":  "Trent Jones",
    "email":  "trent.jones@example.com",
    "phone":  "503-555-1221",
    "items": [
      {
        "itemNo": 321,
        "description":  "Product-21",
        "quantity": 3,
        "unitPrice": 49.95,
        "totalPrice": 149.85
      }
    ],
    "totalPrice": 149.85
  }
]
```

Example Success Response

```
Response-payload: null

200 OK
```

Error Codes

Status Code	Description
400 BAD REQUEST	Returned if one or more of the supplied keys is an invalid format.
404 NOT FOUND	Returned if the region is not found.
414 REQUEST URI TOO LONG	Returned if the URI is longer than the system component can handle. Limiting the size to 2000 bytes will work for most components.
500 INTERNAL SERVER ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception.

PUT /gemfire-api/v1/{region}/{key}?op=REPLACE

This topic describes the PUT command in VMware Tanzu GemFire to update (replace) data with keys if and only if the keys exists in a region. The keys must be present in the region for the update to occur.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/{region}/{key}?op=REPLACE
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/{region}/{key1},{key2},...{keyN}?op=REPLACE
```

Parameters

Parameter	Description	Example Values
op	When you specify REPLACE for this parameter, data is only updated if the specified key or keys are already present in the region.	REPLACE
@type	Specified in the response body. Use this to declare the domain object type of the entry value.	com.mycompany.ObjectName

Example Request

```
Request Payload: application/json
PUT //gemfire-api/v1/orders/2?op=REPLACE

Accept: application/json
Content-Type: application/json
{
    "@type":  "org.apache.geode.web.rest.domain.Order",
    "purchaseOrderNo": 1121,
    "customerId": 1012,
    "description":  "Order for XYZ Corp",
    "orderDate":  "02/10/2020",
    "deliveryDate":  "02/20/2020",
    "contact":  "Jelly Bean",
    "email":  "jelly.bean@example.com",
    "phone":  "01-2048096",
    "totalPrice": 225,
    "items": [
      {
          "itemNo": 1,
          "description":  "Product-100",
          "quantity": 12,
          "unitPrice": 5,
          "totalPrice": 60
      }
   ]
}
```

Example Success Response

```
Response Payload: null

200 OK
```

Error Codes

Status Code	Description
400 BAD REQUEST	Returned if the supplied key is not present in the region.
404 NOT FOUND	Returned if the region is not found.
500 INTERNAL SERVER ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception.

PUT /gemfire-api/v1/{region}/{key}?op=CAS

This topic describes the PUT command in VMware Tanzu GemFire to update (compare-and-set) value-having key with a new value if and only if the @old value sent matches the current value-having key in the region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/{region}/{key}?op=CAS
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/{region}/{key1},{key2},...{keyN}?op=CAS
```

Parameters

Parameter	Description	Example Values
op	URL parameter. When you specify CAS for this parameter, data is only updated if the @old value specified in the request body matches the existing value in the region.	CAS
@type	Specified in the response body for both the old and new value. Use this to declare the domain object type of the entry's value.	com.mycompany.ObjectName

@old	Compare this value to the existing value in the region.	{
		"@type": "org.apache.geode.web.rest.domain.Order",
		"purchaseOrderNo": 1121,
		"customerId": 1012,
		"description": "Order for XYZ Corp",
		"orderDate": "02/10/2020",
		"deliveryDate": "02/20/2020",
		"contact": "Jelly Bean",
		"email": "jelly.bean@example.com",
		"phone": "01-2048096",
		"items": [
		{
		"itemNo": 1,
		"description": "Product-100",
		"quantity": 12,
		"unitPrice": 5,
		"totalPrice": 60
		}
		],
		"totalPrice": 225
		}

Parameter	Description	Example Values

@new	If @old value matches existing value, use this value to replace the existing value.	{
		"@type": "org.apache.geode.web.rest.domain.Order",
		"purchaseOrderNo": 1121,
		"customerId": 1013,
		"description": "Order for New Corp",
		"orderDate": "02/10/2020",
		"deliveryDate": "02/25/2020",
		"contact": "Vanilla Bean",
		"email": "vanilla.bean@example.com",
		"phone": "01-2048096",
		"items": [
		{
		"itemNo": 12345,
		"description": "part 123",
		"quantity": 12,
		"unitPrice": 29.99,
		"totalPrice": 149.95
		}
		],
		"totalPrice": 149.95
		}

Example Request

```
Request Payload: application/json

PUT /gemfire-api/v1/orders/2?op=CAS

Accept: application/json
Content-Type: application/json
{
   "@old": {
       "@type":  "org.apache.geode.web.rest.domain.Order",
       "purchaseOrderNo": 1121,
       "customerId": 1012,
       "description":  "Order for  XYZ Corp",
       "orderDate":  "02/10/2020",
       "deliveryDate":  "02/20/2020",
       "contact":  "Jelly Bean",
       "email":  "jelly.bean@example.com",
       "phone":  "01-2048096",
       "items": [
         {
             "itemNo": 1,
             "description":  "Product-100",
             "quantity": 12,
             "unitPrice": 5,
             "totalPrice": 60
         }
      ],
       "totalPrice": 225
   },
    "@new ": {
       "@type":  "org.apache.geode.web.rest.domain.Order",
       "purchaseOrderNo": 1121,
       "customerId": 1013,
       "description":  "Order for  New Corp",
       "orderDate":  "02/10/2020",
       "deliveryDate":  "02/25/2020",
       "contact":  "Vanilla Bean",
       "email":  "vanillabean@example.com",
       "phone":  "01-2048096",
       "items": [
         {
             "itemNo": 12345,
             "description":  "part 123",
             "quantity": 12,
             "unitPrice": 29.99,
             "totalPrice": 149.95
         }
      ],
       "totalPrice": 149.95
   }
}
```

Example Success Response

```
Response Payload: null

200 OK
```

Error Codes

Status Code	Description
400 BAD REQUEST	Returned if the supplied key is not present in the region.
404 NOT FOUND	Returned if the region is not found.
409 CONFLICT	Returned if the provided @old value of the key does not match the current value of the key.
500 INTERNAL SERVER ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception.

Example Error Response

```
Response-payload: application/json

409 Conflict
Content-Type: application/json
{
    "purchaseOrderNo": 1121,
    "customerId": 1012,
    "description":  "Order for  XYZ Corp",
    "orderDate":  "02/10/2020",
    "deliveryDate":  "02/20/2020",
    "contact":  "Jelly Bean",
    "email":  "jelly.bean@example.com",
    "phone":  "01-2048096",
    "items": [
      {
          "itemNo": 1,
          "description":  "Product-100",
          "quantity": 12,
          "unitPrice": 5,
          "totalPrice": 60
      }
   ],
    "totalPrice": 225
}
```

Implementation Notes

If the “@old” value sent by the client in the HTTP request, along with the “@new” value, does not match the existing value having key in region, then a 409 - CONFLICT error is returned indicating the mismatch in expected state. The “@old” and current value must match in order for the key to be assigned the “@new” value.

If a “CONFLICT” occurs, it is a simple matter for the client to issue a HTTP GET request for the Key (GET /gemfire-api/v1/orders/222) to get a updated copy of the value. CAS is similar to optimistic locking (as opposed to optimistic locking assuming the value will change between the time a client requests a value and subsequently updates the value) in that it assumes the client’s state is up-to-date when the client tries to update, but if not then fail, hence the 409 - CONFLICT.

DELETE /gemfire-api/v1/{region}

This topic describes the DELETE command in VMware Tanzu GemFire to delete all entries in the region. This is limited to replicated regions only, and is not available for partitioned regions.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/{region}
```

Parameters

None.

Example Request

```
Request Payload: null
DELETE /gemfire-api/v1/orders
Accept: application/json
```

Example Success Response

```
Response Payload: null
200 OK
```

Error Codes

Status Code	Description
404 NOT FOUND	Returned if the region is not found.
500 INTERNAL SERVER ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception.

DELETE /gemfire-api/v1/{region}/{key}

This topic describes the DELETE command in VMware Tanzu GemFire to delete the entry for the specified key in the region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/{region}/{key}
```

Parameters

None.

Example Request

```
Request Payload: null
DELETE /gemfire-api/v1/orders/11
Accept: application/json
```

Example Success Response

```
Response Payload: null
200 OK
```

Error Codes

Status Code	Description
404 NOT FOUND	Returned if the region or specified key is not found.
500 INTERNAL SERVER ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception.

DELETE /gemfire-api/v1/{region}/{key1},{key2},…{keyN}

This topic describes the DELETE command in VMware Tanzu GemFire to delete entries for multiple keys in the region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/{region}/{key1},{key2},...{keyN}
```

Parameters

None.

Example Request

```
Request Payload: null
DELETE /gemfire-api/v1/orders/12,13
Accept: application/json
```

Example Success Response

```
Response Payload: null
200 OK
```

Error Codes

Status Code	Description
404 NOT FOUND	Returned if either the region or one or more of the specified keys is not found.
500 INTERNAL SERVER ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception.

Query Endpoints

This topic provides links to information about VMware Tanzu GemFire query endpoints.

Tanzu GemFire uses a query syntax based on OQL (Object Query Language) to query region data. Since Tanzu GemFire regions are key-value stores, values can range from simple byte arrays to complex nested objects.

- GET /gemfire-api/v1/queries

  List all parameterized queries by ID or name.

- POST /gemfire-api/v1/queries?id=<queryId>&q=<OQL-statement>

  Create (prepare) the specified parameterized query and assign the corresponding ID for lookup.
- POST /gemfire-api/v1/queries/{queryId}

  Execute the specified named query passing in scalar values for query parameters in the POST body.
- PUT /gemfire-api/v1/queries/{queryId}

  Update a named, parameterized query.
- DELETE /gemfire-api/v1/queries/{queryId}

  Delete the specified named query.
- GET /gemfire-api/v1/queries/adhoc?q=<OQL-statement>

  Run an unnamed (unidentified), ad-hoc query passed as a URL parameter.

GET /gemfire-api/v1/queries

This topic describes the GET command in VMware Tanzu GemFire to list all parameterized queries by ID or name.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/queries
```

Parameters

None

Example Request

```
Request Payload: null
GET /gemfire-api/v1/queries/
Accept: application/json
```

Example Response

```
Response Payload: application/json

200 OK
Content-Type: application/json
Location: http://localhost:8080/gemfire-api/v1/queries

{
   "queries": [
      {
         "id":  "selectCustomer",
         "oql":  "SELECT c FROM /customers c WHERE c.customerId = $1"
      },
      {
         "id":  "selectHighRollers",
         "oql":  "SELECT DISTINCT c FROM /customers c, /orders o WHERE o.totalprice > $1 AND c.customerId = o.customerI
d"
      }
   ]
}
```

Error Codes

Status Code	Description
401 UNAUTHORIZED	Invalid Username or Password
403 FORBIDDEN	Insufficient privileges for operation
500 INTERNAL SERVER ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception.

POST /gemfire-api/v1/queries?id=&q=

This topic describes the PUT command in VMware Tanzu GemFire to create or prepare the specified parameterized query and assign the corresponding ID for lookup.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/queries?id=<queryId>&q="<OQL-statement
>"
```

Parameters

Parameter	Description	Example Values
id	Required. Unique identifier for the named parameterized query.	selectCustomers
q	Required. OQL query statement. Use double quotes to surround the OQL query statement.	"SELECT o FROM /orders o WHERE o.quantity > $1 AND o.totalprice > $2"

For this release, you cannot specify the query string inside the request body (as JSON).

You must specify the query as a URL parameter.

Example Request

```
POST /gemfire-api/v1/queries?id=selectOrders&q="SELECT o FROM /orders o WHERE o.quantity > $1 AND o.totalprice > $2"
Accept: application/json
```

Example Success Response

```
Response Payload: null

201 CREATED
Location: http://localhost:8080/gemfire-api/v1/queries/selectOrders
```

Error Codes

Status Code	Description
400 BAD REQUEST	Query ID not specified or malformed OQL statement
401 UNAUTHORIZED	Invalid Username or Password
403 FORBIDDEN	Insufficient privileges for operation
409 CONFLICT	QueryId already assigned to another query
500 INTERNAL SERVER ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception. • Query store does not exist!

POST /gemfire-api/v1/queries/{queryId}

This topic describes the POST command in VMware Tanzu GemFire to execute the specified named query passing in scalar values for query parameters in the POST body.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/queries/{queryId}
```

Parameters

Parameter	Description	Example Values
{queryId}	QueryID for named query.	selectOrders

Parameter	Description	Example Values

query bind parameter values	Bind parameters for the query are specified in the request body (JSON).	Specify the parameter @type and @value for each bind parameter. For example, to provide values to the following query:
		`SELECT o FROM /orders o WHERE o.quantity > $1 AND o.totalprice > $2`
		You could pass in the following JSON in the request body as the bind parameters:
		`[ { "@type": "int ", "@value": 2 }, { "@type": "double ", "@value": 110.00 } ]`

Example Request

```
POST /gemfire-api/v1/queries/selectOrders
Accept: application/json
Content-Type: application/json

[
      {
          "@type":  "int ",
          "@value": 2
      },
      {
         "@type":  "double ",
          "@value": 110.00
      }
]
```

Example Success Response

```
Response Payload: application/json

200 OK
Content-Length: <#-of-bytes>
Content-Type: application/json

[
  {
    "description":  "Purchase order for  company - B",
     "totalPrice": 350,
     "purchaseOrderNo": 1112,
     "customerId": 102,
     "deliveryDate":  "Thu Feb 20 00:00:00 IST 2020",
     "contact":  "John Doe",
     "email":  "John.Doe@vmware.com",
     "phone":  "01-2048096",
     "items": [
       {
           "description":  "Product-AAAA",
           "quantity": 10,
           "itemNo": 1,
           "unitPrice": 20,
           "totalPrice": 200,
           "type-class":  "org.apache.geode.web.rest.domain.Item"
       },
       {
           "description":  "Product-BBB",
           "quantity": 15,
           "itemNo": 2,
           "unitPrice": 10,
           "totalPrice": 150,
           "type-class":  "org.apache.geode.web.rest.domain.Item"
       }
    ],
     "orderDate":  "Mon Feb 10 00:00:00 IST 2020",
     "type-class":  "org.apache.geode.web.rest.domain.Order"
  },
  {...},
  {...}
}
```

Error Codes

Status Code	Description
400 BAD REQUEST	JSON document in the request body (specifying query bind params) is invalid
401 UNAUTHORIZED	Invalid Username or Password
403 FORBIDDEN	Insufficient privileges for operation
404 NOT FOUND	Query with specified ID could not be found

Status Code	Description
500 INTERNAL SERVER ERROR	Encountered error at server: • Syntax of the OQL queryString is invalid! • A function was applied to a parameter that is improper for that function! • Bind parameter is not of the expected type! • Name in the query cannot be resolved!" • The number of bound parameters does not match the number of placeholders! • Query is not permitted on this type of region! • Query execution time has exceeded max query execution time (gemfire.Cache.MAX_QUERY_EXECUTION_TIME) configured! • Data referenced in from clause is not available for querying!" • Query execution gets canceled due to low memory conditions and the resource manager critical heap percentage has been set!" • Error encountered while executing named query!"

PUT /gemfire-api/v1/queries/{queryId}

This topic describes the PUT command in VMware Tanzu GemFire to update a named, parameterized query.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/queries/{queryId}
```

Parameters

Parameter	Description	Example Values
q	OQL String. The OQL query must be a SELECT expression that references at least one region.	"SELECT DISTINCT from /customers WHERE lastName=$1"

> You cannot specify the query string inside the request body as JSON. You must specify the query as a URL parameter.

Example Request

```
PUT /gemfire-api/v1/queries/selectOrders?q="SELECT DISTINCT from /customers where lastName=$1"

Accept: application/json
Content-Length: <#-of-bytes>
```

Example Success Response

```
Response Payload:  null

200 OK
```

Error Codes

Status Code	Description
401 UNAUTHORIZED	Invalid Username or Password
403 FORBIDDEN	Insufficient privileges for operation
404 NOT FOUND	QueryID does not exist
500 INTERNAL SERVER ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception.

Implementation Notes

This operation is idempotent, meaning multiple identical requests should have the same effect as the initial request.

DELETE /gemfire-api/v1/queries/{queryId}

This topic describes the DELETE command in VMware Tanzu GemFire to delete the specified named query.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/queries/{query}
```

Parameters

Parameter	Description	Example Values
{queryId}	QueryID for named query to delete.	selectOrders

Example Request

```
DELETE /gemfire-api/v1/queries/selectOrders

Accept: application/json
Content-Type: application/json
```

Example Success Response

```
Response Payload: application/json

200 OK
```

Error Codes

Status Code	Description
401 UNAUTHORIZED	Invalid Username or Password
403 FORBIDDEN	Insufficient privileges for operation
404 NOT FOUND	Query with specified ID could not be found
500 INTERNAL SERVER ERROR	Encountered error at server. Check the HTTP response body for a stack trace of the exception.

GET /gemfire-api/v1/queries/adhoc?q=

This topic describes the GET command in VMware Tanzu GemFire to run an unnamed (unidentified) ad-hoc query passed as a URL parameter.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/queries/adhoc?q=<OQL-statement>
```

Parameters

Parameter	Description	Example Values
q	Required. OQL query statement. : Since the query string is passed in the URL, the OQL must be URL-encoded. Some HTTP clients such as Web browsers will automatically encode URLs; however, if you are not using one of those clients, you will need to URL encode the query string yourself.	SELECT o FROM /orders o WHERE o.quantity > 2 AND o.totalprice > 110.00 (or URL encoded: SELECT%20o%20FROM%20%2Forders%20o%20WHERE%20o.quantity%20%3E%202%20AND%20o.totalprice%20%3E%20110.00) SELECT * FROM /customers (or URL encoded: SELECT%20*%20FROM%20/customers)

Example Request

```
curl -i "http://localhost:8080/gemfire-api/v1/queries/adhoc?q=select%20*%20%20from%20/customers"
```

Example Success Response

```
Response Payload: application/json

200 OK
Content-Length: <#-of-bytes>
Content-Type: application/json
[
   {
      "firstName":  "John",
       "lastName":  "Doe",
       "customerId": 101,
   },
   {
       "firstName":  "Jane",
       "lastName":  "Doe",
       "customerId": 102,
   },
   {
     ....
   }
]
```

Error Codes

Status Code	Description
401 UNAUTHORIZED	Invalid Username or Password
403 FORBIDDEN	Insufficient privileges for operation
500 INTERNAL SERVER ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception. Some possible exceptions include: • A function was applied to a parameter that is improper for that function! • Bind parameter is not of the expected type! • Name in the query cannot be resolved! • The number of bound parameters does not match the number of placeholders! • Query is not permitted on this type of region! • Query execution time is exceeded max query execution time (gemfire.Cache.MAX_QUERY_EXECUTION_TIME) configured! • Data referenced in from clause is not available for querying! • Query execution gets canceled due to low memory conditions and the resource manager critical heap percentage has been set! • Server has encountered an error while executing Adhoc query!

Function Endpoints

This topic provides links to information about VMware Tanzu GemFire function endpoints. Tanzu GemFire functions allow you to write and execute server-side transactions and data operations. These transactions

and operations may include anything ranging from initializing components or third-party services or aggregating data.

- GET /gemfire-api/v1/functions

  List all registered Tanzu GemFire functions in the cluster.

- POST /gemfire-api/v1/functions/{functionId}

  Execute Tanzu GemFire function on entire cluster or on a specified region, members and member groups.

GET /gemfire-api/v1/functions

This topic describes the GET command in VMware Tanzu GemFire to list all registered Tanzu GemFire functions in the cluster.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/functions
```

Parameters

None.

Example Request

```
GET /gemfire-api/v1/functions
Accept: application/json
```

Example Success Response

```
Response Payload: application/json

200 OK
Content-Length: <#-of-bytes>
Content-Type: application/json
Location: https://localhost:8080/gemfire-api/v1/functions

{
  "functions": [
    "AddFreeItemToOrders",
    "GetRegions",
    "GetDeliveredOrders"
  ]
}
```

Error Codes

Status Code	Description
404 NOT FOUND	Returned if no functions are found in the cluster.
500 INTERNAL SERVER ERROR	Error encountered at Tanzu GemFire server. Check the HTTP response body for a stack trace of the exception.

POST /gemfire-api/v1/functions/{functionId}

This topic describes the POST command in VMware Tanzu GemFire to execute a Tanzu GemFire function on an entire cluster, on a specified region, or on members or member groups.

Resource URL

```
/gemfire-api/v1/functions/{functionId}?[&onRegion=regionname|&onMembers=member1,member2,...,memberN|&onGroups=group1,group2,...,groupN]
```

Parameters

- {functionId} This required parameter is the name of the function to execute. Place it in the resource URL, as in the example request: AddFreeItemToOrders.
- onRegion This optional parameter specifies the target region for the function. You can only invoke a function on a single region. Substitute the region's name for regionname within the sample syntax onRegion=regionname.
- onMembers This optional parameter specifies the target members of the function. For multiple members, specify a comma-delimited list of member names, as in the sample onMembers=member1,member2.
- onGroups This optional parameter specifies the target groups of the function. For multiple groups, specify a comma-delimited list of group names, as in the sample onGroups=membergroup1,membergroup2.
- filter This optional parameter can only be used with the onRegion parameter, where the region has a data-policy of PARTITION. The parameter specifies a list of applicable keys that the function needs to filter on. There are three keys in the example resource URL:

  ```
  http://serverURL/functions/SampleFunction?onRegion=TestPartitionRegion&filter=key1,key2,key3
  ```

- Any *function arguments* are passed in the request body in JSON format. The content of the arguments depends on how the function is defined. Each function argument must be written as a JSON object (enclosed in braces { }) and using @type to declare its type. @value can be used to specify a scalar value. Nested JSON objects ({ }) or JSON collections ([ ]) are not accepted. Collections of Java objects cannot be specified as parameters in JSON format, so if a function expects a collection of objects as one of its arguments (for example, function(List<Item> list)), it cannot be called using the Tanzu GemFire REST API. Following are some examples of functions and their arguments in JSON format:

Function signature	Function arguments in JSON format

myFunction1(int n)	[
	{
	"@type": "integer",
	"@value": 10
	}
	]

Function signature	Function arguments in JSON format

myFunction2(double d, Item i)	[
	{
	"@type": "double",
	"@value": 210
	},
	{
	"@type": "org.apache.geode.web.rest.domain.Item",
	"itemNo": "599",
	"description": "Part X Free on Bumper Offer",
	"quantity": "2",
	"unitprice": "5",
	"totalprice": "10.00"
	}
	]
	
	Notice how item fields are written in JSON format due to the restrictions against nested objects. The following approach would be incorrect:
	
	[
	{
	"@type": "double",
	"@value": 210
	},
	{
	"@type": "org.apache.geode.web.rest.domain.Item",
	"itemNo" : {
	"@type":"double",
	"@value": 599
	},
	"description": "Part X Free on Bumper Offer",
	"quantity": "2",
	"unitprice": "5",
	"totalprice": "10.00"
	}
	]

Example Requests

```
Request Payload: application/json
POST /gemfire-api/v1/functions/AddFreeItemToOrders
Accept: application/json
Content-Type: application/json

[
      {
          "@type":  "double",
          "@value": 210
      },
      {
          "@type":  "org.apache.geode.web.rest.domain.Item",
          "itemNo":  "599",
          "description":  "Part X Free on Bumper Offer",
          "quantity":  "2",
          "unitprice":  "5",
          "totalprice":  "10.00"
      }
]
```

Another example:

```
Request Payload: null

POST /gemfire-api/v1/functions/getDeliveredOrders
Accept: application/json
```

Example Success Responses

```
Response Payload: null
200 OK
Location:http: //localhost:8080/gemfire-api/v1/functions/AddFreeItemToOrders
```

Another example response:

```
Response Payload: application/json

200 OK
Content-Length: 316
Content-Type: application/json
Location: http://localhost:8080/gemfire-api/v1/functions/getDeliveredOrders
[
   {
       "purchaseOrderNo":  "1121",
       "deliveryDate":  "Thu Oct 10 00:00:00 IST 2019"
   },
   {
       "purchaseOrderNo":  "777",
       "deliveryDate":  "Thu Oct 10 00:00:00 IST 2019"
   },
   {
      ...
   }
]
```

Error Codes

Status code 500 INTERNAL SERVER ERROR is an error encountered in a server. Check the HTTP response body for a stack trace of the exception.

Administrative Endpoints

This topic provides links to information about VMware Tanzu GemFire administrative endpoints. Administrative endpoints provide management and monitoring functionality for the REST API interface.

- [HEAD | GET] /gemfire-api/v1/ping

  Mechanism to check for REST API server and service availability.

- GET /gemfire-api/v1/servers

  Mechanism to obtain a list of all members in the cluster that are running the REST API service.

[HEAD | GET] /gemfire-api/v1/ping

This topic describes the HEAD GET command in VMware Tanzu GemFire to check for REST API server and service availability.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/ping
```

Parameters

None.

Example Request

```
HEAD /gemfire-api/v1/ping
GET /gemfire-api/v1/ping
```

Example Success Response

```
200 OK
```

Error Codes

Status Code	Description
404 NOT FOUND	The Developer REST API service is not available.
500 INTERNAL SERVER ERROR	Encountered error at server. Check the Tanzu GemFire exception trace.

GET /gemfire-api/v1/servers

This topic describes the GET command in VMware Tanzu GemFire to obtain a list of all members in the cluster that run the REST API service.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/gemfire-api/v1/servers
```

Parameters

None.

Example Request

```
Request Payload: null
Request: GET /gemfire-api/v1/servers
Response Payload: application/json
```

Example Success Response

```
200 OK

Content-Length: <#-of-bytes>
Content-Type: application/json; charset=utf-8

[
 "http://<HOST_NAME1>:<PORT1>",
 "http://<HOST_NAME2>:<PORT2>",
 "http://<HOST_NAME3>:<PORT3>",
 "http://<HOST_NAME4>:<PORT4>"
]
```

Error Codes

Status Code	Description
500 INTERNAL SERVER ERROR	Returned if Tanzu GemFire throws an error while executing the request.

Tools, Modules, and Extensions

This topic provides links to documentation that describes and explains tools, modules, and extension associated with VMware Tanzu GemFire.

- gfsh (GemFire SHell)

  gfsh (pronounced “jee-fish”) provides a single, powerful command-line interface from which you can launch, manage, and monitor Tanzu GemFire processes, data, and applications.

- Gemcached

  Gemcached is deprecated and will be removed in upcoming releases.

  Gemcached is a Tanzu GemFire adapter that allows Memcached clients to communicate with a Tanzu 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.

- Tanzu GemFire Session Management

  The VMware Tanzu GemFire HTTP Session Management extensions provide fast, scalable, and reliable session replication for HTTP servers without requiring application changes.

- Pulse

  Pulse is a Web Application that provides a graphical dashboard for monitoring vital, real-time health and performance of Tanzu GemFire clusters, members, and regions.

  > GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

- Visual Statistics Display (VSD)

  The Visual Statistics Display utility reads Tanzu GemFire statistics and produces graphical displays for analysis.

gfsh (GemFire SHell)

This topic provides links to documentation about gfsh, the VMware Tanzu GemFire command-line interface.

gfsh (pronounced “jee-fish”) provides a single, powerful command-line interface from which you can launch, manage, and monitor Tanzu GemFire processes, data, and applications.

- What You Can Do with gfsh

  gfsh supports the administration, debugging, and deployment of VMware Tanzu GemFire processes and applications.

- Starting gfsh

  Before you start gfsh, confirm that you have set JAVA_HOME and that your PATH variable includes the gfsh executable.

- Configuring the gfsh Environment

  The gfsh.bat and gfsh bash script automatically append the required VMware Tanzu GemFire and JDK .jar libraries to your existing class path. There are user-configurable properties you can set for security, environment variables, logging, and troubleshooting.

- Useful gfsh Shell Variables

  You can use the built-in gfsh shell variables in scripts.

- Basic Shell Features and Command-Line Usage

  The gfsh 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.

- Specifying JSON within Command-Line Options

  Some gfsh commands allow JSON specifications within command-line options.

- Tutorial—Performing Common Tasks with gfsh

  This topic takes you through a typical sequence of tasks that you execute after starting gfsh.

- Quick Reference of gfsh Commands by Functional Area

  This quick reference sorts all commands into functional areas.

- gfsh Command Help

  This section provides help and usage information about all gfsh commands, listed alphabetically.

- Creating and Running gfsh Command Scripts

  gfsh offers several ways to run commands in a scripting environment.

- Running gfsh Commands on the OS Command Line

- Mapping of cache.xml Elements to gfsh Configuration Commands.

  You can configure a Tanzu GemFire cluster using either cache.xml files, or you can use gfsh and the cluster configuration service to configure a cluster. This section maps cache.xml elements to the gfsh commands that configure and manage a cluster.

What You Can Do with gfsh

This topic describes what you can do with gfsh, the VMware Tanzu GemFire command-line interface.

gfsh supports the administration, debugging, and deployment of GemFire processes and applications.

With gfsh, you can:

- Start and stop Tanzu 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 Tanzu GemFire processes

- Launch Tanzu 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, 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.

You can connect gfsh to a remote cluster using the HTTP protocol. For more information about this, see Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS.

By default, the cluster configuration service saves the configuration of your GemFire cluster when you create GemFire objects using gfsh. You can export this configuration and import it into another GemFire cluster. For more information about this, see Overview of the Cluster Configuration Service.

Starting gfsh

This topic explains how to start gfsh, the VMware Tanzu GemFire command-line interface.

> Before you try to start gfsh, confirm that you have set the JAVA_HOME environment variable and that your PATH variable includes the gfsh executable.
>
> On Windows, you must have set the JAVA_HOME environment variable to use gfsh start, stop, and status commands for both locators and servers.

To launch gfsh, execute the following command on a machine that has Tanzu GemFire installed:

Windows:

```
<product_directory>\bin\gfsh.bat
```

Where *<product_directory>* corresponds to the location where you installed Tanzu GemFire.

Unix:

```
<product_directory>/bin/gfsh
```

Where *<product_directory>* corresponds to the location where you installed Tanzu GemFire. Upon execution, the gfsh script appends the required Tanzu GemFire and JDK Jar libraries to your existing class path.

If you have successfully started gfsh, the gfsh splash screen and prompt appears.

```
c:\gemfire\latest>gfsh.bat
    _________________________     __
   / _____/ _______/ _______/ /____/ /
  / /  __/ /___  /______  / ______  /
 / /__/ / ____/  ______/ / /    / /
/_______/_/      /_______/_/    /_/

Monitor and Manage GemFire
gfsh>
```

You can also run some gfsh commands directly within your terminal without entering a gfsh prompt. For example:

Windows:

```
prompt> gfsh start server --name=server1
```

Unix:

```
$ gfsh start server --name=server1
```

For more information, see Creating and Running gfsh Command Scripts.

Start gfsh with a Custom Wrapper Script

You can gfsh using a custom wrapper shell script. If you have a script created for a previous version of Tanzu GemFire, update the script to use the new gemfire-bootstrap.jar. When you create a script, do not include any other jars on the java -classpath argument other than the gemfire-bootstrap.jar. VMware recommends that you avoid wildcards or other globbing patterns that may unintentionally include other jars. Use the --automatic-module-path bootstrap argument to include any custom classes at startup.

For an example script, see the run-gfsh script in the gemfire-examples repository in GitHub.

Configuring the gfsh Environment

This topic explains how to configure the VMware Tanzu GemFire command-line interface (gfsh) environment.

You can set user-configurable properties for security, environment variables, logging, and troubleshooting.

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 for details on these security properties.

Configuring gfsh Environment Variables

In addition, you can set 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_QUIET_EXECUTION

See Useful gfsh Shell Variables for more information.

Configuring gfsh Session Logging

By default, gfsh session logging is deactivated. To enable gfsh logging, you must set the Java system property -Dgfsh.log-level=desired_log_level where *desired_log_level* is one of the following values: severe, warning, info, config, fine, finer, finest. For example, in Linux:

```
export JAVA_ARGS=-Dgfsh.log-level=info
```

Then, start gfsh.

gfsh produces a log file named gfsh-*%u_%g*.log. This log file records the events of an individual gfsh session. It includes environment information, such as Java and system information, and detailed command execution. The variables are replaced as follows:

- %u - a unique number to resolve conflicts
- %g - the generation number to distinguish rotated logs

gfsh uses the JDK Logger to generate gfsh session log files. For a description of how the variables are used in naming the log file, see Class FileHandle on the Oracle documentation website. The default name of the generated gfsh log file cannot be changed.

By default, the log file is written to the current working directory where you have executed the gfsh or gfsh.bat script. To modify the directory location where log files are written, use the gfsh.log-dir Java system property. For example:

```
export JAVA_ARGS="-Dgfsh.log-level=info -Dgfsh.log-dir=/machinename/logs"
```

Then, start gfsh.

In addition, gfsh records a history of commands in the ${SYS_USER_HOME}/.geode/.gfsh.history file, which you can use to create scripts or review past commands.

Member Log Files

gfsh writes several log files for any members that are started by using gfsh. Useful member log files include:

- <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.

Viewing Standard Output and Standard Error

By default, GemFire does not show messages written by the application to standard output and standard error. To allow these messages to be written to the locator and server log files, respectively, specify the --redirect-output option with the gfsh start locator or start server commands. For example, the following command causes stdout and stderr messages to be written to the locator1.log file:

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

Tab Completion

This section applies only to UNIX installations.

When you run gfsh commands from a UNIX bash shell, you can enable automatic tab-completion in the shell by running the following command:

```
source <gemfire-install-directory>/bin/gfsh-completion.bash
```

After running this command, you can use auto completion when running gfsh commands from the bash shell.

See Using Tab Completion.

Command History and gfsh.history

A history of commands that have been executed successfully is logged in .gfsh.history file in a .gemfire directory under 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.

Formatting of Results

This section applies only to UNIX installations.

gfsh commands such as query produce output with wide columns that may become misaligned and require manual reformatting to view the output. If the output cannot fit in the available width of the terminal, gfsh automatically trims the columns widths to fit. You can deactivate this behavior by setting the gfsh environment variable APP_RESULT_VIEWER to an arbitrary value that is not external or basic.

You can set the APP_RESULT_VIEWER variable to external to enable viewing of the output using the UNIX less command.

See Configuring gfsh Environment Variables.

Useful gfsh Shell Variables

This topic discusses pre-defined and custom shell variables in gfsh, the VMware Tanzu GemFire command-line interface.

You can use the pre-defined 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 Tanzu 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_SEPARATOR	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_FETCH_SIZE	Fetch size to be used while querying. Values: 0 - 2147483647. Default value is 100.
APP_LAST_EXIT_STATUS	Last command exit status. Similar to $? (Unix) and %errorlevel% (Windows). Values: 0 (successful), 1 (error), 2 (installation failure).
APP_LOGGING_ENABLED	Whether gfsh logging is enabled. Default: false (read only). You can enable gfsh logging by setting the gfsh.log-level Java system property to a supported Java log level.
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.
APP_QUIET_EXECUTION	Whether the execution should be in quiet mode. Values (case insensitive): true, false. Default value is false.

APP_RESULT_VIEWER	Unix only. Set this variable to external to enable viewing of the output using the UNIX less command. Default value is basic (gfsh).

Basic Shell Features and Command-Line Usage

This topic explains the basic shell features and command-line usage for gfsh, the VMware Tanzu GemFire command-line interface.

The gfsh utility provides useful features for a shell environment, including command auto-completion, preserved command history, and delimiting of multi-line commands. Context-sensitive help is available by command and by topic.

To view a list of available gfsh commands, press Tab at an empty prompt.

The list of commands you see depends on whether you are connected to a Tanzu GemFire cluster. If you are not connected, you see a list of local commands that are available.

Use the hint command to get information about a particular topic.

The hint command displays a one-line description and associated commands for a specified topic. For example, 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 Tanzu GemFire distributed system.

clear defined indexes: Clears all the defined indexes.
create defined indexes: Creates all the defined indexes.
create index: Create an index that can be used when executing queries.
define index: Define 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.
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.
```

To view a list of hint topics, type hint.

Use the help command to get information about 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, and depend on the connection status of gfsh. The following example assumes gfsh is not currently connected (via connection to a JMX Manager node), so some start commands are unavailable.

```
gfsh>help start
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.
```

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
```

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
```

Hitting tab after start locator will populate the --name parameter after the command. (If you do not specify a member name, gfsh will automatically pick a random name. This is useful for automation.)

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 a backslash.

When entering long commands, you can break the command line up by using the backslash character (’\’) as a delimiter. For example:

```
gfsh>create region --name=region1 \
--type=REPLICATE_PERSISTENT \
--cache-writer=org.apache.geode.examples.MyCacheWriter \
--group=Group1 --disk-store=DiskStore1
```

Enclose strings that contain spaces or commas with single or double quotes

When you execute gfsh commands in the gfsh command shell, enclose any strings that contain spaces or commas in single quotes. For example:

```
gfsh>start locator --name='locator 1'
```

```
start locator --name=locator1 --port=9009 \
 --J='-Dgemfire.remote-locators=192.0.2.0[9009],192.0.2.1[9009]'
```

When you execute multiple gfsh commands in a single line from the operating system shell, enclose the gfsh commands in double quotes. Within the double quotes, enclose any strings that contain spaces or commas with single quotes. For example:

```
$ gfsh -e "start locator --name='locator 1'" -e "start server --name=server1"
```

Tutorial—Performing Common Tasks with gfsh

This topic describes a typical sequence of tasks that you execute after starting gfsh, the VMware Tanzu GemFire command-line interface.

Step 1: Create a scratch working directory and navigate to that directory. For example:

```
$ mkdir gfsh_tutorial
$ cd gfsh_tutorial
```

Step 1: Start a gfsh prompt.

```
$ gfsh
    _________________________     __
   / _____/ _______/ ______/ /____/ /
  / /  __/ /___  /______  / _____  /
 / /__/ / ____/  ______/ / /    / /
/_______/_/      /______/_/    /_/

Monitor and Manage Tanzu GemFire
gfsh>
```

See Starting gfsh for details.

Step 2: Start up a locator. Enter the following command:

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

The following output appears:

```
gfsh>start locator --name=locator1
Starting a GemFire Locator in /home/username/gfsh_tutorial/locator1...
.....................
Locator in /home/username/gfsh_tutorial/locator1 on 192.168.100.102[10334] as locator1 is currently online.
Process ID: 96393
Uptime: 12 seconds
Geode Version: 10.1.0
Java Version: 11.0.10
Log File: /home/username/gfsh_tutorial/locator1/locator1.log
JVM Arguments: --add-exports=java.management/com.sun.jmx.remote.security=ALL-UNNAMED --add-exports=java.base/sun.ni
o.ch=ALL-UNNAMED --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.nio=ALL-UNNAMED -Dge
mfire.enable-cluster-configuration=true -Dgemfire.load-cluster-configuration-from-dir=false -Dgemfire.launcher.registerSignalHan
dlers=true -Djava.awt.headless=true -Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /home/username/gfsh_tutorial/lib/gemfire-bootstrap-10.1.0.jar

Successfully connected to: JMX Manager [host=192.168.100.102, port=1099]

Cluster configuration service is up and running.
```

If you run start locator from gfsh without specifying the member name, gfsh automatically provides a random member name. This is useful for automation.

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 3: 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 2, the following output appears:

```
gfsh>describe connection
Connection Endpoints
----------------------
192.168.100.102[1099]
```

Notice that the JMX Manager is on port 1099 while the locator was assigned the default port of 10334.

Step 4: 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 3 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 2.

```
gfsh>connect
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=192.168.100.102, port=1099] ..
Successfully connected to: [host=192.168.100.102, port=1099]

You are connected to a cluster of version 10.1.0.
```

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=hostname[port] when connecting to the cluster. For example (type disconnect first if you want to try this next command):

```
gfsh>connect --locator=localhost[10334]
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=192.168.100.102, port=1099] ..
Successfully connected to: [host=192.168.100.102, port=1099]

You are connected to a cluster of version 10.1.0.
```

Another way to connect gfsh to the cluster 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=localhost[1099]
Connecting to Manager at [host=localhost, port=1099] ..
Successfully connected to: [host=localhost, port=1099]

You are connected to a cluster of version 10.1.0.
```

In addition, you can connect to remote clusters over HTTP. See Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS.

Step 5: Disconnect and close the second terminal window. Type the following commands to disconnect and exit the second gfsh prompt:

```
gfsh>disconnect
Disconnecting from: localhost[1099]
Disconnected from : localhost[1099]
gfsh>exit
```

Close the second terminal window.

Step 6: Start a server. Return to your first terminal window, and start a cache server that uses the locator that you started in Step 2. Enter the following command:

```
gfsh>start server --name=server1 --locators=localhost[10334]
```

If the server starts successfully, the following output appears:

```
gfsh>start server --name=server1 --locators=localhost[10334]
Starting a GemFire Server in /home/username/gfsh_tutorial/server1...
...
Server in /home/username/gfsh_tutorial/server1 on 192.168.100.102[40404] as server1 is currently online.
Process ID: 96851
Uptime: 3 seconds
Geode Version: 10.1.0
Java Version: 11.0.10
Log File: /home/username/gfsh_tutorial/server1/server1.log
JVM Arguments: --add-exports=java.management/com.sun.jmx.remote.security=ALL-UNNAMED --add-exports=java.base/sun.ni
o.ch=ALL-UNNAMED --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.nio=ALL-UNNAMED -Dge
mfire.start-dev-rest-api=false -Dgemfire.locators=localhost[10334] -Dgemfire.use-cluster-configuration=true -XX:OnOutOfMemor
yError=kill -KILL %p -Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true -Dsun.rmi.dgc.server.gcInterval=
9223372036854775806
Class-Path: /home/username/gfsh_tutorial/lib/gemfire-bootstrap-10.1.0.jar
```

If you run start server from gfsh without specifying the member name, gfsh automatically provides a random member name. This is useful for automation.

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 log files.

Step 7: List members. Use the list members command to view the current members of the cluster you have just created.

```
gfsh>list members
Member Count : 2

  Name   |                    Id                     |  Type   | Status
-------- | ---------------------------------------------------- | ------- | ------
locator1 | 192.168.100.102(locator1:96393:locator)<ec><v0>:463.. | Locator | Ready
server1  | 192.168.100.102(server1:96851)<v1>:43740             | Server  | Ready
```

Step 8: View member details by executing the describe member command.

```
gfsh>describe member --name=server1
Name            : server1
Id              : 192.168.100.102(server1:96851)<v1>:43740
Type            : Server
Host            : 192.168.100.102
Regions         :
Timezone        : America/Los_Angeles -08:00
Metrics URL     : Not Available
PID             : 96851
Groups          :
Redundancy-Zone :
Used Heap       : 32M
Max Heap        : 8192M
Load Average1   : 3.22
Working Dir     : /home/username/gfsh_tutorial/server1
Log file        : /home/username/gfsh_tutorial/server1/server1.log
Locators        : localhost[10334]

Cache Server Information
Server Bind             :
Server Port             : 40404
Running                 : true

Client Connections : 0
```

No regions have been assigned to this member yet.

Step 9: Create your first region. Enter the following command followed by the tab key:

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

A list of possible region types appears:

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

LOCAL                                LOCAL_HEAP_LRU
LOCAL_OVERFLOW                       LOCAL_PERSISTENT
LOCAL_PERSISTENT_OVERFLOW            PARTITION
PARTITION_HEAP_LRU                   PARTITION_OVERFLOW
PARTITION_PERSISTENT                 PARTITION_PERSISTENT_OVERFLOW
PARTITION_PROXY                      PARTITION_PROXY_REDUNDANT
PARTITION_REDUNDANT                  PARTITION_REDUNDANT_HEAP_LRU
PARTITION_REDUNDANT_OVERFLOW         PARTITION_REDUNDANT_PERSISTENT
PARTITION_REDUNDANT_PERSISTENT_OVERFLOW   REPLICATE
REPLICATE_HEAP_LRU                   REPLICATE_OVERFLOW
REPLICATE_PERSISTENT
REPLICATE_PERSISTENT_OVERFLOW
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 | Message
------- | ------ | ---------------------------------------
server1 | OK     | Region "/region1" created on "server1"

Cluster configuration for group 'cluster' is updated.
```

Because only one server is in the cluster at the moment, the command creates the local region on server1.

Step 10: 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 GemFire Server in /home/username/gfsh_tutorial/server2...
...
Server in /home/username/gfsh_tutorial/server2 on 192.168.100.102[40405] as server2 is currently online.
Process ID: 97094
Uptime: 3 seconds
Geode Version: 10.1.0
Java Version: 11.0.10
Log File: /home/username/gfsh_tutorial/server2/server2.log
JVM Arguments: --add-exports=java.management/com.sun.jmx.remote.security=ALL-UNNAMED --add-exports=java.base/sun.ni
o.ch=ALL-UNNAMED --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.nio=ALL-UNNAMED -Dge
mfire.default.locators=192.168.100.102[10334] -Dgemfire.start-dev-rest-api=false -Dgemfire.use-cluster-configuration=true -XX:
OnOutOfMemoryError=kill -KILL %p -Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true -Dsun.rmi.dgc.se
rver.gcInterval=9223372036854775806
Class-Path: /home/username/gfsh_tutorial/lib/gemfire-bootstrap-10.1.0.jar
```

Step 11: Create a replicated region.

```
gfsh>create region --name=region2 --type=REPLICATE
Member  | Status | Message
------- | ------ | ---------------------------------------
server1 | OK     | Region "/region2" created on "server1"
server2 | OK     | Region "/region2" created on "server2"

Cluster configuration for group 'cluster' is updated.
```

Step 12: Create a partitioned region.

```
gfsh>create region --name=region3 --type=PARTITION
Member  | Status | Message
------- | ------ | ---------------------------------------
server1 | OK     | Region "/region3" created on "server1"
server2 | OK     | Region "/region3" created on "server2"

Cluster configuration for group 'cluster' is updated.
```

Step 13: Create a replicated, persistent region.

```
gfsh>create region --name=region4 --type=REPLICATE_PERSISTENT
Member  | Status | Message
------- | ------ | ---------------------------------------
server1 | OK     | Region "/region4" created on "server1"
server2 | OK     | Region "/region4" created on "server2"

Cluster configuration for group 'cluster' is updated.
```

Step 14: List regions. A list of all the regions you just created displays.

```
gfsh>list regions
List of regions
---------------
region1
region2
region3
region4
```

Step 15: View member details again by executing the describe member command.

```
gfsh>describe member --name=server1
Name           : server1
Id           : 192.168.100.102(server1:96851)<v1>:43740
Type           : Server
Host           : 192.168.100.102
Regions        : region1
                 region2
                 region3
                 region4
Timezone       : America/Los_Angeles -08:00
Metrics URL    : Not Available
PID            : 96851
Groups         :
Redundancy-Zone :
Used Heap      : 58M
Max Heap       : 8192M
Load Average1  : 2.27
Working Dir    : /home/username/gfsh_tutorial/server1
Log file       : /home/username/gfsh_tutorial/server1/server1.log
Locators       : localhost[10334]

Cache Server Information
Server Bind            :
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.

> The Metrics URL specifies the location of your metrics module. If metrics are not enabled, or if there errors occurred starting the metrics endpoint, the Metrics URL will show as Not Available.

```
gfsh>describe member --name=server2
Name           : server2
Id             : 192.168.100.102(server2:97094)<v2>:43723
Type           : Server
Host           : 192.168.100.102
Regions        : region1
                 region2
                 region3
                 region4
Timezone       : America/Los_Angeles -08:00
Metrics URL    : Not Available
PID            : 97094
Groups         :
Redundancy-Zone :
Used Heap      : 57M
Max Heap       : 8192M
Load Average1  : 2.03
Working Dir    : /home/username/gfsh_tutorial/server2
Log file       : /home/username/gfsh_tutorial/server2/server2.log
Locators       : 192.168.100.102[10334]

Cache Server Information
Server Bind              :
Server Port              : 40405
Running                  : true

Client Connections : 0
```

Note that even though you brought up the second server after creating the first region (region1), the second server still lists region1 because it picked up its configuration from the cluster configuration service.

```
gfsh>describe member --name=locator1
Name           : locator1
Id             : 192.168.100.102(locator1:96393:locator)<ec><v0>:46363
Type           : Locator
Host           : 192.168.100.102
Regions        :
Timezone       : America/Los_Angeles -08:00
Metrics URL    : Not Available
PID            : 96393
Groups         :
Redundancy-Zone :
Used Heap      : 160M
Max Heap       : 8192M
Load Average1  : 2.08
Working Dir    : /home/username/gfsh_tutorial/locator1
Log file       : /home/username/gfsh_tutorial/locator1/locator1.log
Locators       : 192.168.100.102[10334]
```

Step 16: 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 17: 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 18: 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    | ubuntu(server2:6092)<v2>:17443
server1    | ubuntu(server1:5931)<v1>:35285
```

Notice that because the entry was put into a replicated region, the entry is located on both cluster 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 19: Export your data. To save region data, you can 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.

Step 20: Shutdown the cluster.

```
gfsh>shutdown --include-locators=true
```

Quick Reference of gfsh Commands by Functional Area

This topic provides links to other topics about VMware Tanzu GemFire command-line interface (gfsh) commands, sorted into functional areas.

- Basic VMware Tanzu GemFire gfsh Commands
- Configuration Commands
- Data Commands
- Deployment Commands
- Disk Store Commands
- Durable CQ and Client Commands
- Function Execution Commands
- Gateway (WAN) Commands
- GemFire AsynEventQueue Commands
- GemFire Monitoring Commands
- Index Commands
- JMX Connection Commands
- Locator Commands
- PDX Commands
- Query Service Commands
- Region Commands
- Server Commands

Basic GemFire gfsh Commands by Functional Area

This topic lists basic VMware Tanzu GemFire command-line interface (gfsh) commands, sorted into functional areas.

Command	Description	Availability
debug	Activate or deactivate debugging output in gfsh.	online, offline
echo	Echo the given text, which may include system and user variables.	online, offline
exit	Exit the gfsh shell. You can also use quit to exit the shell.	online, offline
help	If the argument is a gfsh command, displays syntax and usage information for the command. If there are no arguments, displays a list of all available commands.	online, offline
hint	Display information about topics and a list of commands associated with a topic.	online, offline
history	Show or save the command history.	online, offline
run	Execute a set of gfsh commands.	online, offline

Command	Description	Availability
set variable	Set variables in the gfsh environment.	online, offline
sh	Execute operating system (OS) commands.	offline, online
sleep	Delay gfsh command execution.	online, offline
version	Display product version information.	online, offline

Configuration Commands

Command	Description	Availability
alter runtime	Alter configuration properties for a specific member or members while the member or members are running.	online
change loglevel	Change the logging level on specified servers.	online
configure pdx	Alter cluster-wide PDX configuration settings for all caches. Note: This command must be run before starting data members in order to enforce configuration settings.	online
describe config	Display the configuration of a member.	online
export config	Export configurations, data, logs and stack-traces.	online
export cluster-configuration	Export a shared configuration ZIP file that contains cache.xml files, gemfire.properties files and JAR files needed to configure and operate a GemFire cluster.	online
import cluster-configuration	Import an exported configuration.	online
status cluster-config-service	Report on the status of the cluster configuration server.	online

Data Commands

Command	Description	Availability
export data	Export user data from a region to a file.	online
get	Display an entry in a region.	online
import data	Import user data from a file to a region.	online
locate entry	Locate a region entry on a member.	online
put	Add or update a region entry.	online
query	Run queries against GemFire regions.	online
remove	Remove an entry from a region.	online
wan-copy region	Copy the entries of a region in a WAN site onto the same region in another WAN site, using a gateway sender.	online

Deployment Commands

Command	Description	Availability
deploy	Deploy JAR-packaged applications to a member or members. Note: When deploying a JAR, the name of the JAR cannot contain any of the following characters: *, !, #, @, ?.	online
list deployed	Display a list of JARs that were deployed to members using the deploy command.	online
undeploy	Undeploy the JAR files that were deployed on members or groups using the deploy command.	online

Disk Store Commands

Command	Description	Availability
alter disk-store	Modify an existing GemFire resource.	online
backup disk-store	Back up persistent data from all members to the specified directory.	online
compact disk-store	Compact an online disk store on all members with that disk store.	online
compact offline-disk-store	Compact an offline disk store.	online, offline
create disk-store	Define a pool of one or more disk stores, which can be used by regions and client subscription queues.	online
describe disk-store	Display information about a member's disk store.	online
describe offline-disk-store	Display information about an offline member's disk store	online, offline
destroy disk-store	Delete a disk store and all files on disk used by the disk store. Data for closed regions that previously used this disk store is lost.	online
list disk-stores	List all available disk stores in a GemFire cluster.	online
revoke missing-disk-store	Instruct the members of a distributed system to stop waiting for a disk store to be available.	online
show missing-disk-stores	Display a summary of the disk stores that are currently missing from a distributed system.	online
upgrade offline-disk-store	Upgrade offline disk stores created prior to 10.1 to segmented disk stores.	offline
validate offline-disk-store	Validate offline disk stores.	online, offline

Durable CQ and Client Commands

Command	Description	Availability
list durable-cqs	List durable client CQs associated with the specified durable client id.	online
close durable-cq	Close the durable CQ registered by the durable client and drain events held for the durable CQ from the subscription queue.	online
close durable-client	Attempt to close the durable client. The client must be disconnected.	online
show subscription-queue-size	Show the number of events in the subscription queue. If a CQ name is provided, count the number of events in the subscription queue for the specified CQ.	online

Function Execution Commands

Command	Description	Availability
destroy function	Destroy or unregister a function. The default is for the function to be unregistered from all members.	online
execute function	Execute the function with the specified ID. By default, executes on all members.	online
list functions	Display a list of registered functions. The default is to display functions for all members.	online

Gateway (WAN) Commands

Command	Description	Availability
alter async-event-queue	Alter attributes of a specified asynchronous event queue.	online
create async-event-queue	Create an asynchronous event queue.	online
create gateway-receiver	Create a gateway receiver on one or more members.	online
create gateway-sender	Create a gateway sender on one or more members.	online
describe async-event-queue	Display details about an asynchronous event queue created in the cluster members.	online
describe gateway-receiver	Display details about a gateway receiver created in the cluster members.	online
describe gateway-sender	Display details about a gateway sender created in the cluster members.	online
destroy async-event-queue	Destroy an asynchronous event queue on one or more members.	online
destroy gateway-sender	Destroy a gateway sender on one or more members.	online
list async-event-queues	Display a list of async event queues for all members.	online
list gateways	Display the gateway senders and receivers for a member or members.	online
load-balance gateway-sender	Cause the specified gateway sender to close its current connections and reconnect to remote gateway receivers in a more balanced fashion.	online
pause gateway-sender	Pause a gateway sender.	online
resume async-event-queue-dispatcher	Resume dispatching of events on a specified asynchronous event queue which is currently paused.	online
resume gateway-sender	Resume any gateway senders that you have paused.	online
start gateway-receiver	Start a gateway receiver on a given member or group of members.	online
start gateway-sender	Start a gateway sender on a member or members.	online
status gateway-receiver	Display the status of the specified gateway receiver.	online
status gateway-sender	Display the status of the specified gateway sender.	online
stop gateway-receiver	Stop a gateway receiver on a member or members.	online
stop gateway-sender	Stop a gateway sender with a given id on a specified member or members of a specified member group.	online

GemFire AsyncEventQueue Commands

Command	Description	Availability
alter async-event-queue	Alter attributes of a specified asynchronous event queue.	online
create async-event-queue	Create an asynchronous event queue.	online
destroy async-event-queue	Destroy an asynchronous event queue on one or more members.	online
list async-event-queues	Display a list of async event queues for all members.	online
resume async-event-queue-dispatcher	Resume dispatching of events on a specified asynchronous event queue which is currently in a paused state.	online

> You can configure multiple regions to share the same AsyncEventQueue if they use the same AsyncEventListener. The maximum combined number of AsyncEventQueues and GatewaySenders per cluster is 128.

GemFire Monitoring and Managing Commands

Command	Description	Availability
cancel query	Cancel a currently executing query.	online
describe client	Display details about a specified client.	online
describe member	Display details of a member with given name/id.	online
describe query	Display details about an executing query.	online
export logs	Export/dump logs to a given directory.	online
export stack-traces	Export the stack trace for a member or members.	online
gc	Force garbage collection on a member or members.	online
list clients	Display a list of connected clients.	online
list members	Display all or a subset of members.	online
list queries	Display a list of queries currently running in the cluster.	online
netstat	Report network information and statistics via the “netstat” operating system command.	online
show dead-locks	Display deadlocks, logs, metrics and missing disk-stores.	online
show log	Display the log for a member.	online
show metrics	Display or export metrics for the entire distributed system, a member or a region.	online
show statistics	Display statisitcs from one or more members.	online
shutdown	Shut down all members that have a cache.	online
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	online, offline
start jvisualvm	Start the JDK’s Java VisualVM monitoring application in a separate process.	online, offline

Command	Description	Availability
start pulse	Launch the GemFire Pulse monitoring dashboard tool in the user's default system browser.	online, offline
start vsd	Launch GemFire Visual Statistics Display (VSD) in a separate process.	online, offline

Index Commands

Command	Description	Availability
clear defined indexes	Clear all the defined indexes.	online, offline
create defined indexes	Create all the defined indexes.	online
create index	Create an index that can be used when executing queries.	online
define index	Define an index that can be used when executing queries. Then you can create multiple indexes all at once.	online, offline
destroy index	Destroy or remove the specified index.	online
list indexes	Display the list of indexes created for all members.	online

JMX Connection Commands

Command	Description	Availability
connect	Connect to a jmx-manager either directly or via a locator.	offline
describe connection	Display connection information details.	online, offline
disconnect	Close any active connections.	online

Locator Commands

Command	Description	Availability
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 automatically starts an embedded JMX Manager and connects the current gfsh session to the JMX Manager.	online, offline
status locator	Displays the status of the specified locator.	online, offline
stop locator	Stop a locator.	online, offline

PDX Commands

Command	Description	Availability
configure pdx	Configure Portable Data eXchange for all the caches in the cluster.	online, offline
pdx rename	Rename PDX types in an offline disk store.	online, offline

Region Commands

Command	Description	Availability
alter region	Alter the configuration of a region.	online
create region	Create and configure a region.	online
describe region	Display the attributes and key information of a region.	online
destroy region	Destroy or remove a region.	online
list regions	Display regions of a member or members. If no parameter is specified, all regions in the GemFire distributed system are listed.	online
rebalance	Rebalance partitioned regions.	online
restore redundancy	Restore redundancy to partitioned regions and optionally reassign which members host the primary copies.	online
status redundancy	Display the redundancy status of partitioned regions.	online

Server Commands

Command	Description	Availability
start server	Start a GemFire cache server process.	online, offline
status server	Display the status of the specified GemFire cache server.	online, offline
stop server	Stop a GemFire cache server.	online, offline

gfsh Command Help

This topic links to help and usage documentation for VMware Tanzu GemFire command-line interface (gfsh) commands. Each topic provides additional information about the command, including syntax, a list of options, and examples.

- alter

  Modify an existing Tanzu GemFire resource.

- backup disk-store

  Back up persistent data from all members to the specified directory.

- cancel

  Cancel a currently executing query.

- change loglevel

  Changes the logging level on specified members.

- clear defined indexes

  Clears all the defined indexes.

- close

  Close durable client CQs and durable clients.

- compact

Compact online and offline disk-stores.

- configure

  Configure Portable Data eXchange for all the caches in the cluster.

- connect

  Connect to a jmx-manager either directly or via a locator.

- create

  Create async-event-queues, disk-stores, gateway receivers, gateway senders, indexes, and regions.

- debug

  Activate or deactivate debugging output in gfsh.

- define index

  Define an index that can be used when executing queries. Then, you can execute a single command to create multiple indexes all at once using create defined indexes.

- deploy

  Deploy JAR-packaged applications to a member or members.

  > When deploying a JAR, the name of the JAR cannot contain any of the following characters: *, !, #, @, ?.

- deregister driver

  Deregister a driver with the cluster's Driver Manager using the name of a driver class contained within a currently deployed jar.

- describe

  Display details of a member's configuration, shell connection, disk-stores, members, or regions.

- destroy

  Delete or unregister functions, remove indexes, disk stores, and regions.

- disconnect

  Close any active connections.

- echo

  Echo the given text, which may include system and user variables.

- execute function

  Execute functions on members or regions.

- exit

  Exit the gfsh shell. You can also use quit to exit the shell.

- export

  Export configurations, data, logs and stack-traces.

- gc

  Force GC (Garbage Collection) on a member or members.

- get

  Display an entry in a region.

- help

  Display syntax and usage information for all the available commands.

- hint

  Display information about topics and a list of commands associated with a topic.

- history

  Show or save the command history.

- import

  You can import data into a region or import an existing cluster configuration into the cluster.

- list

  List existing Tanzu GemFire resources such as deployed applications, disk-stores, functions, members, servers, and regions.

- load-balance gateway-sender

  Causes the specified gateway sender to close its current connections and reconnect to remote gateway receivers in a more balanced fashion.

- locate entry

  Locate a region entry on a member.

- netstat

  Report network information and statistics via the “netstat” operating system command.

- pause gateway-sender

  Pause a gateway sender.

- pdx rename

  Renames PDX types in an offline disk store.

- put

  Add or update a region entry.

- query

  Run queries against Tanzu GemFire regions.

- rebalance

  Rebalance partitioned regions.

- register driver Register a driver with the cluster’s Driver Manager using the name of a driver class contained within a currently deployed jar.

- remove

  Remove an entry from a region.

- restore redundancy

  Restore redundancy to partitioned regions and optionally reassign which members host the primary copies.

- resume

Resume a paused asynch event dispatcher or gateway sender.

- revoke missing-disk-store

  Instruct the members of a distributed system to stop waiting for a disk store to be available.

- run

  Execute a set of GFSH commands.

- set variable

  Set variables in the GFSH environment.

- sh

  Execute operating system commands.

- show

  Display deadlocks, logs, metrics and missing disk-stores.

- shutdown

  Stop all members.

- sleep

  Delay gfsh command execution.

- start

  Start servers, locators, gateway senders and gateway receivers, and monitoring tools.

- status

  Check the status of the cluster configuration service and Tanzu GemFire member processes, including locators, gateway receivers, gateway senders, and servers.

- stop

  Stop gateway receivers, gateway senders, locators and servers.

- undeploy

  Undeploy the JAR files that were deployed on members or groups using deploy command.

- upgrade

  Upgrade offline disk stores from the format used in GemFire releases prior to 10.1 to the segmented disk store type introduced in GemFire version 10.1.

- validate offline-disk-store

  Validate offline disk stores.

- version

  Display product version information.

- wan-copy region

  Copy the data of a region from a WAN site to the same region on another WAN site by using a gateway sender.

alter

This topic describes the alter command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to modify an existing VMware Tanzu GemFire resource.

- alter async-event-queue

  Modifies attributes of an async event queue

- alter disk-store

  Modifies or removes a region from an offline disk-store.

- alter gateway-sender

  Modifies attributes of a gateway-sender.

- alter query-service

  Alter configuration details of the query configuration service.

- alter region

  Alters the configuration of a region.

- alter runtime

  Alters configuration properties for all members or a subset of members while the member or members are running.

alter async-event-queue

Alter attributes of a specified asynchronous event queue. Each server hosting the specified async event queue must be restarted for the new attribute settings to take effect on that server.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
alter async-event-queue --id=value [--batch-size=value] [--batch-time-interval=value]
    [--max-queue-memory=value] [--if-exists(=value)] [--pause-event-processing(=value)]
```

The required option, --id, identifies the async event queue to be altered.

Parameters, alter async-event-queue

Name	Description
--id	*Required.* ID of the async event queue to be changed
--batch-size	Maximum number of events that a batch can contain
--batch-time-interval	Maximum amount of time, in ms, that can elapse before a batch is delivered
--max-queue-memory	Maximum amount of memory, in megabytes, that the queue can consume before overflowing to disk
--if-exists	If the specified async event queue does not exist, gfsh responds with a message to that effect. If this parameter is true, the response is prefixed with the label "Skipping: ". Useful for scripted tests. Default if the parameter is not specified: false. Default (if the parameter is specified without value): true.
--pause-event-processing	Specifies whether event dispatching from the queue to the listeners will be paused when the AsyncEventQueue is started. Default if the parameter is not specified: false. Default if the parameter is specified without value: true.

Example Commands:

```
alter async-event-queue --id=myAsyncEventQueue --batch-size=50 --if-exists
```

alter disk-store

Modify or remove a region from an offline disk-store.

When modifying a region's configuration, it is customary to take the region off-line and restart using the new configuration. You can use the alter disk-store command to change the configuration of the region stored in the disk-store to match the configuration you will use at restart.

Availability: Offline.

Syntax:

```
alter disk-store --name=value --region=value --disk-dirs=value(,value)*
   [--compressor(=value)] [--concurrency-level=value]
   [--enable-statistics=value] [--initial-capacity=value] [--load-factor=value]
   [--lru-algorithm=value] [--lru-action=value] [--lru-limit=value]
   [--off-heap(=value)] [--remove(=value)]
```

The three required options, --name, --region, and --disk-dirs, identify the disk store and region to be altered. If no additional options are specified, gfsh displays the current configuration without making any changes.

Parameters, alter disk-store

Name	Description
--name	*Required*. Name of the disk-store whose contents will be altered.
--region	*Required*. Name (including path) of the region using the disk store.
--disk-dirs	*Required*. Directories where the data for the disk store was previously written.
--compressor	The fully-qualified class name of the compressor to use when compressing region entry values. A value of none removes the compressor.
--concurrency-level	An estimate of the maximum number of application threads that will concurrently access a region entry. Together with --initial-capacity and --load-factor, sets the parameters on the underlying java.util.ConcurrentHashMap used for storing region entries. This attribute does not apply to partitioned regions.
--enable-statistics	Enables statistics for the region specified by the --region option. Valid values are true or false. If the parameter is specified without a value, the value defaults to true.
--initial-capacity	Together with --concurrency-level and --load-factor, sets the parameters on the underlying java.util.ConcurrentHashMap used for storing region entries.
--load-factor	Together with --concurrency-level and --initial-capacity, sets the parameters on the underlying java.util.ConcurrentHashMap used for storing region entries. This must be a floating point number between 0 and 1, inclusive.

Name	Description
--lru-action	Action to take when evicting entries from the region. Valid values are: • none • overflow-to-disk • local-destroy
--lru-algorithm	Least recently used eviction algorithm. Valid types are: • none • lru-entry-count • lru-heap-percentage • lru-memory-size
--lru-limit	Number of entries allowed in the region before eviction occurs.
--off-heap	Specifies whether the region values are in heap memory or off-heap memory. When true, region values are in off-heap memory. If the parameter is specified without a value, the value defaults to true.
--remove	Specifies whether to remove the region from the disk-store. If the parameter is specified without a value, the value defaults to true. Note: --remove deletes all persistent data for the region. You might want to copy the disk store files to a backup before using this option to allow you to retrieve the data at a later time.

Example Commands:

```
alter disk-store --name=DiskStore1 --region=region1 --disk-dirs=/Disks/DiskStore1 --off-heap
alter disk-store --name=DiskStore1 --region=region1 --disk-dirs=/Disks/DiskStore1 --remove
```

alter gateway-sender

Alter attributes of a specified gateway sender on one or more members of a cluster.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
alter gateway-sender --id=value [--members=value(,value)*] [--groups=value(,value)*]
   [--alert-threshold=value] [--batch-size=value] [--batch-time-interval=value]
   [--gateway-event-filter=value(,value)*]
```

The required option, --id, identifies the gateway sender to be altered.

Parameters, alter gateway-sender

Name	Description
--id	*Required*. ID of the gateway sender to be changed
--members	Name or ID of the members whose configuration is to be altered at runtime. If you do not specify this parameter, the configuration properties are modified for all cluster members using the cluster configuration service.

Name	Description
--groups	Name of the groups whose members' runtime configurations are to be altered. If you do not specify this parameter, the configuration properties are modified for all cluster members using the cluster configuration service.
--alert-threshold	Maximum time, in milliseconds, that a region event can remain in the gateway sender queue before Tanzu GemFire logs an alert.
--batch-size	Maximum number of messages that a batch can contain.
--batch-time-interval	Maximum amount of time, in ms, that can elapse before a batch is delivered, when no events are found in the queue to reach the batch-size.
--gateway-event-filter	A list of fully-qualified class names of GatewayEventFilters (separated by commas) to be associated with the GatewaySender. This serves as a callback for users to filter out events before dispatching to a remote cluster. For example: `gateway-event-filter=com.user.filters.MyFilter1,com.user.filters.MyFilters2` In case no value is provided, all existing filters will be removed.

Example Commands:

```
alter gateway-sender --id=sender1 --alert-threshold=100 --batch-size=200
```

alter query-service

Alter configuration details of the query configuration service.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
alter query-service --method-authorizer=value
    [--authorizer-parameters=value(;value)*] [--force-update(=value)]
```

Parameters, alter query-service

Name	Description
--method-authorizer	*Required.* Fully qualified class name of the MethodInvocationAuthorizer to be used for query authorization.
--authorizer-parameters	A semicolon-separated list of parameters to be used by the specified MethodInvocationAuthorizer. This requires that a method-authorizer option has been specified.
--force-update	Specifies whether to forcibly update the MethodInvocationAuthorizer, even when there are continuous queries registered in the member. Default (if the parameter is not specified): false. Default (if the parameter is specified without value): true. Note: when set as true, any registered CQ will pick up the new MethodInvocationAuthorizer and invalidate its internal cache; consider checking that the new MethodInvocationAuthorizer allows the methods invoked by the CQs before using this option.

Example Commands:

```
alter query-service --method-authorizer=org.apache.geode.cache.query.security.UnrestrictedMethodAuthorizer
alter query-service --method-authorizer=org.apache.geode.cache.query.security.UnrestrictedMethodAuthorizer --force-update=true
alter query-service --method-authorizer=org.apache.geode.cache.query.security.JavaBeanAccessorMethodAuthorizer --authorizer-parameters=java.lang;java.util
```

alter region

Alters the configuration of a region.

See Specifying JSON within Command-Line Options for syntax details.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
alter region --name=value [--groups=value(,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)?]
  [--entry-time-to-live-reset-policy(=value)?]
  [--entry-idle-time-custom-expiry=value] [--entry-time-to-live-custom-expiry=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)?]
  [--cache-listener=value(,value)*] [--cache-loader=value]
  [--cache-writer=value] [--async-event-queue-id=value(,value)*]
  [--gateway-sender-id=value(,value)*] [--enable-cloning(=value)?]
  [--eviction-max(=value)?]
```

Parameters, alter region

Name	Description	Default Value
--async-event-queue-id	IDs of the Async Event Queues that will be used for write-behind operations.	
--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. A fully qualified class name may be appended with a JSON specification that will be parsed to become the fields of the parameter to the init() method for a class that implements the Declarable interface.	
--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. A fully qualified class name may be appended with a JSON specification that will be parsed to become the fields of the parameter to the initialize() method for a class that implements the Declarable interface.	
--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. A fully qualified class name may be appended with a JSON specification that will be parsed to become the fields of the parameter to the init() method for a class that implements the Declarable interface.	

Name	Description	Default Value
--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.	false
--entry-idle-time-expiration	Number of seconds before a region or an entry expires. Specify -1 to indicate that there is no expiration of this type	-1
--entry-idle-time-expiration-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 cluster. 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 cluster. 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
--entry-time-to-live-expiration	Number of seconds before a region or an entry expires. Specify -1 to indicate that there is no expiration of this type.	-1
--entry-time-to-live-expiration-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 cluster. 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 cluster. 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
--entry-time-to-live-reset-policy	Determines if the entry's timeout should be relative to its creation time or the last modified time. Valid options are never and last-modified.	last-modified
--entry-idle-time-custom-expiry	The name of a class implementing CustomExpiry for entry idle time. Append a JSON string for initialization properties.	

Name	Description	Default Value
--entry-time-to-live-custom-expiry	The name of a class implementing CustomExpiry for entry time to live. Append a JSON string for initialization properties.	
--eviction-max	Maximum value for the Eviction Attributes that the eviction algorithm uses to determine when to perform its eviction action. The unit of the maximum value is determined by the Eviction Algorithm.	0
--gateway-sender-id	IDs of the Gateway Senders where data is routed.	
--groups	Groups of members where the region will be altered.	
--name	Required. Name (including path) of the region.	
--region-idle-time-expiration	Deprecated.Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	-1
--region-idle-time-expiration-action	Deprecated.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 cluster. 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 cluster. 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
--region-time-to-live-expiration	Deprecated.Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	-1
--region-time-to-live-expiration-action	Deprecated.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 cluster. 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 cluster. 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 Commands:

```
alter region --name=region1 --eviction-max=5000 [-group=all]
```

Sample Output:

```
gfsh>alter region --name=customer --eviction-max=5000
Member  | Status
------- | ---------------------------------
server1 | Region "/customer" altered on "server1"
```

alter runtime

Alters configuration properties for all servers or a subset of servers while the member or members are running. Alter runtime is a cluster configuration command that takes immediate effect on the existing members of the cluster, as well as on any new members added to the cluster.

The alter runtime command does not apply to locators.

For more information about these configuration properties, see cache.xml and configuration parameter reference.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
alter runtime [--members=value(,value)*] [--groups=value(,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] [--copy-on-read(=value)?] [--lock-lease=value]
   [--lock-timeout=value] [--message-sync-interval=value] [--search-timeout=value]
```

Parameters, alter runtime

Name	Description	Default Value
--members	Name or ID of the members whose configuration is to be altered at runtime. If you do not specify this parameter, the configuration properties are modified for all cluster members using the cluster configuration service.	If not specified, all members using the cluster configuration service
--groups	Name of the groups whose members's runtime configuration is to be altered. If you do not specify this parameter, the configuration properties are modified for all cluster members using the cluster configuration service.	If not specified, all members using the cluster configuration service
--archive-disk-space-limit	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. Valid values are (in megabytes): 0 to 1000000.	0
--archive-file-size-limit	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. Valid values are (in megabytes): 0 to 1000000.	0

Name	Description	Default Value
--log-disk-space-limit	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. Valid values are (in megabytes): 0 to 1000000.	0
--log-file-size-limit	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 zero, log rolling is deactivated. Valid values are (in megabytes): 0 to 1000000.	0
--log-level	The new log level. This option is required and you must specify a value. Valid values are: ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF.	INFO
--statistic-archive-file	The file to which the running system member writes statistic samples. For example: "StatisticsArchiveFile.gfs". Must be defined to store the archiving to a file. Adding the .gz suffix to the file name causes it to be compressed. See Statistics.	*not set*
--statistic-sample-rate	Statistic sampling rate. Valid values are (in milliseconds): 1000 to 60000. See Statistics.	1000
--enable-statistics	Deprecated.This parameter is deprecated as statistics collection will be enabled by default. It will be removed in upcoming releases. Whether statistic sampling should be enabled. Specify --statistic-archive-file to store the statistics to a file. Valid values are: true and false. See Statistics.	true
--copy-on-read	True or false. Sets the copy on read region attribute for cache read operations. See Copy on Read Behavior.	false
--lock-lease	Sets the length, in seconds, of distributed lock leases obtained by this cache. See Setting Cache Timeouts.	120
--lock-timeout	Sets the number of seconds a cache operation may wait to obtain a distributed lock lease before timing out. See Setting Cache Timeouts.	60
--message-sync-interval	Sets the frequency (in seconds) at which a message will be sent by the primary cache-server node to all the secondary cache-server nodes to remove the events which have already been dispatched from the queue. See Change Server Queue Synchronization Frequency.	1
--search-timeout	Sets the number of seconds a cache get operation can spend searching for a value. See Setting Cache Timeouts.	300

Example Commands:

```
alter runtime --members=server1 --log-level=WARN --enable-statistics=true
```

Sample Output:

```
gfsh>alter runtime --members=server1 --log-level=WARN --enable-statistics=true
Runtime configuration altered successfully for the following member(s)
192.0.2.0(server1:240)<v1>:64871
```

backup disk-store

This topic describes the backup disk-store command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to 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 operating system copy command is not recommended.

You can also use this command to perform an incremental backup. See Creating Backups for System Recovery and Operational Management for more information about incremental backup.

> When connected to a GemFire version 10.1 or later locator, if members running a GemFire version earlier than 10.1 exist in the cluster, issuing the backup disk-store command results in the following error message: "Backup only supports GemFire 10.1 and later. The following members are running on an earlier GemFire version."
> When connected to a GemFire version 10.0 or earlier locator, the backup disk-store runs successfully even if other locators have been upgraded to 10.1 or later.

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]
```

Table 1. Backup Disk-Store Parameters

Name	Description
--dir	*Required*. Directory to which backup files are written.
--baseline-dir	Directory that contains the baseline backup used for comparison during an incremental backup. 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.

Table 1. Backup Disk-Store Parameters

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:\VMwareGemFire100\Latest\server2\. | 192.0.2.0
server1 | 8dc365bd-c086-4af4-99d0-86b0b521aa04 | c:\VMwareGemFire100\Latest\server1\. | 192.0.2.0
```

cancel

This topic describes the cancel command in gfsh, the VMware Tanzu GemFire command-line interface.

- cancel query

  Cancel a currently executing query.

- cancel wan-copy region

  Cancel a wan-copy region command.

---

cancel query

Use this command to cancel a currently executing query.

Availability: Online. In gfsh, you must be connected to a JMX Manager member to use this command.

Syntax:

```
cancel query --query-id=queryId
```

Name	Description
--query-id	*Required.* ID of the query. To find a query ID, use the list queries command to display a list of executing queries and their IDs.

Table 1. Cancel Query Parameters

Example Command:

```
cancel query --queryId=faf13b3b-0da1-4b75-885f-2dc1a010f080
```

Sample Output:

```
gfsh> cancel query --queryId=faf13b3b-0da1-4b75-885f-2dc1a010f080
Query cancelled on members: server-3,server-2
```

---

cancel wan-copy region

Use this command to cancel a wan-copy region command. For more information, see wan-copy region.

Syntax

```
cancel wan-copy region --wan-copy-id=[wan-copy-id]
```

Parameter

This command has the following parameter:

```
--wan-copy-id: Specifies the wan-copy-id of the wan-copy region command to be canceled (required).
```

Possible Statuses and Messages:

```
If the specified wan-copy-id exists and the wan-copy has not started:
   Status: OK
   Message: "Execution canceled."
If the specified wan-copy-id exists and the wan-copy is ongoing:
   Status: ERROR
   Message: "Operation canceled before having copied all entries."
If no wan-copy-id is specified and the wan-copy has not started yet:
   Status: OK
   Message: "No running command to be canceled for region ... and sender ..."
```

Example

cancel wan-copy region –wan-copy-id=cc09e6a1-99a7-4515-ba7c-3ec2caa15446

```
Wan-copy-id=cc09e6a1-99a7-4515-ba7c-3ec2caa15446 is canceled
Attempted to copy 4 region(s)

Region Name: region1-east
Sender Id: ny-east

Member      | Status  | Message
------------|---------|-------------------------------------------------------------------------------
server-ln-1 | OK      | No running command to be canceled for region region1-east and sender ny-east
server-ln-2 | OK      | No running command to be canceled for region region1-east and sender ny-east

Region Name: region1-west
Sender Id: ny-west

Member      | Status  | Message
------------|---------|-------------------------------------------------------------------------------
server-ln-1 | OK      | No running command to be canceled for region region1-west and sender ny-west
server-ln-2 | OK      | No running command to be canceled for region region1-west and sender ny-west

Region Name: region1-north
Sender Id: ny-north

Member      | Status  | Message
------------|---------|--------------------
server-ln-1 | OK      | Execution canceled
server-ln-2 | OK      | Execution canceled

Region Name: region1-south
Sender Id: ny-south

Member      | Status  | Message
------------|---------|--------------------
server-ln-1 | OK      | Execution canceled
server-ln-2 | OK      | Execution canceled
```

change loglevel

This topic describes the change loglevel command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to changes the logging level on specified members. This command takes effect when using the default Tanzu GemFire logging configuration.

When using a custom Log4J configuration, this command takes effect only if the member whose logging level you want to change is started using the --J=-Dgeode.LOG_LEVEL_UPDATE_OCCURS=ALWAYS system property.

The change loglevel command applies only to the members specified, whether they are servers or locators. The change does not apply to unspecified or subsequently-added members.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
change loglevel --loglevel=value [--members=value(,value)*] [--groups=value(,value)*]
```

Table 1. Change Loglevel Parameters

Name	Description	Default Value
--members	Name or ID of one or more members whose logging level you want to change.	
--groups	One or more group names. The logging level changes for all members of these groups.	
--loglevel	*Required.* Log level to change. Valid options are: ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF.	

Table 1. Change Loglevel Parameters

Example Commands:

```
gfsh>change loglevel --loglevel=DEBUG --members=server1
```

Sample Output:

```
gfsh>change loglevel --loglevel=DEBUG --members=server1

Summary

              Member           | Changed loglevel
--------------------------------- | -----------------
192.0.2.0(server1:3060)<v1>:24653 | true
```

clear defined indexes

This topic describes the clear command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to clear all of the defined indexes.

Index definitions are stored locally on the gfsh client. Before creating a new set of indexes or if one or more index creations fail, use this command to clear the definitions.

See also define index.

Availability: Online or offline.

Syntax:

```
clear defined indexes
```

Example Commands:

```
gfsh> clear defined indexes
```

Sample Output:

```
gfsh>clear defined indexes
Index definitions successfully cleared
```

close

This topic describes the close command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to close durable client continuous queries (CQs) and durable clients.

- close durable-client

  Attempts to close a durable client. The client must be disconnected for this command to work.

- close durable-cq

  Closes the durable continuous query (CQ) registered by the durable client and drain events held for the durable CQ from the subscription queue.

close durable-client

Attempts to close a 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 [--members=value(,value)*]
[--groups=value(,value)*]
```

Name	Description
--durable-client-id	*Required.* The ID of the durable client.
--members	Name or ID of the members for which the durable client is to be closed.
--groups	Groups of members for which the durable client is to be closed.

Table 1. Close Durable-Client Parameters

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 continuous query (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
[--members=value(,value)*] [--groups=value(,value)*]
```

Name	Description
--durable-client-id	Required. The ID of the durable client.
--durable-cq-name	Required. Name of the CQ to be closed.
--members	Name or ID of the members for which the durable client is registered and the durable CQ to be closed.
--groups	Groups of members for which the durable client is registered and the durable CQ to be closed.

Table 2. Close Durable-CQ Parameters

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

This topic describes the compact command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to compact online and offline disk stores.

- compact disk-store

  Compact a disk store on all members with that disk store.

- compact offline-disk-store

  Compact an offline disk store.

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 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 [--groups=value(,value)*]
```

Parameters, compact disk-store

Name	Description
--name	*Required.* Name of the disk store to be compacted.
--groups	Groups of members that perform disk compaction. If no group is specified, then the disk store is 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 groups \"{0}\"";
"No members found in the specified groups \"{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, you may need to allocate additional memory to the process by using the --J=-XmxNNNm parameter.

See Running Compaction on Disk Store Log Files for more information.

> 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)*]
```

Parameters, compact offline-disk-store

Name	Description	Default Value
--name	*Required.* Name of the offline disk store to be compacted.	
--disk-dirs	*Required.* One or more directories where data for the disk store was previously written. Separate directories with commas.	
--max-oplog-size	Maximum size (in megabytes) of the oplogs created by compaction.	-1
--J	Arguments passed to the Java Virtual Machine performing the compact operation on the disk store. For example: --J=-Xmx1024m.	

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
```

configure

This topic describes the configure command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to configure Portable Data eXchange (PDX) for all of the caches in the cluster.

configure pdx

This command configures GemFire's Portable Data eXchange for all caches in the cluster and persists the PDX configuration in the locator with the cluster configuration service.

For consistent results, PDX should be configured before any servers have started. A server that is running at the time PDX is configured will not adopt the new configuration until it has been restarted.

Availability: Online.

Syntax:

```
configure pdx [--read-serialized=value] [--ignore-unread-fields=value]
    [--disk-store(=value)?] [--auto-serializable-classes=value(,value)*]
    [--portable-auto-serializable-classes=value(,value)*]
```

Parameters, configure pdx:

Name	Description	Default
--read-serialized	When true, PDX deserialization produces a PdxInstance instead of an instance of the domain class.	false
--ignore-unread-fields	Controls whether PDX ignores fields that were unread during deserialization. The default is to preserve unread fields by including their data during serialization. However, if you configure the cache to ignore unread fields, then their data will be lost during serialization. You should set this attribute to true only if you know this member will only be reading cache data. In this use case you do not need to pay the cost of preserving the unread fields, since you will never be reserializing PDX data.	false
--disk-store	Named disk store where the PDX type data will be stored. If specified without a value, then “DEFAULT” is used.	none
--auto-serializable-classes	Configures ReflectionBasedAutoSerializer as the PDX serializer for member classes. Specifies patterns(a comma separated list) to be matched against domain class names to determine whether they should be auto-serialized. Classes are not checked for portability to non-java languages (equivalent to check-portability=false).	none
--portable-auto-serializable-classes	Configures ReflectionBasedAutoSerializer as the PDX serializer for member classes. Specifies patterns(a comma separated list) to be matched against domain class names to determine whether they should be serialized. Serialization done by the PDX autoserializer will throw an exception if the object of these classes are not portable to non-Java languages (equivalent to check-portability=true).	none

Example Commands:

```
gfsh>configure pdx --read-serialized=true
```

Sample Output:

```
gfsh>configure pdx --read-serialized=true
persistent = false
read-serialized = true
ignore-unread-fields = false

gfsh>configure pdx --disk-store=/home/username/server4/DEFAULT.drf
persistent = true
disk-store = /home/username/server4/DEFAULT.drf
read-serialized = false
ignore-unread-fields = false
```

Error Messages:

```
Configure pdx failed because cluster configuration is deactivated.
```

```
"Failed to persist the configuration changes due to this command, Revert the command to maintain consistency.
Please use "status cluster-config-service" to determining whether Cluster configuration service is RUNNING."
```

connect

This topic describes the connect command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to connect to a JMX manager either directly or through a locator.

If you are connecting via a locator, and a JMX manager does not already exist, the locator starts one.

gfsh connects as a discovery client to the locator service and asks where the JMX Manager is. The locator knows when 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 the locator's JMX RMI port.

You can also connect to a remote locator using the HTTP protocol, as illustrated by the second example below.

You can use this version of gfsh to connect to the JMX manager of Geode version 1.10 or newer. If a command is not supported by the JMX manager to which it is connected, the command will fail with a detailed error message.

Availability: Offline. You will receive a notification "Already connected to: host[port]" if you are already connected.

Syntax:

```
connect [--locator=value] [--jmx-manager=value] [--use-http(=value)?] [--url=value]
    [--username=value] [--password=value] [--token=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)?]
    [--skip-ssl-validation(=value)?]
```

Table 1. Connect Parameters

Name	Description	Default
--locator	Network address of the Locator in the form: host[port].	localhost[10334]
--jmx-manager	Network address of the JMX manager in the form: host[port].	
--use-http	Deprecated: inferred by the presence of --url. Connects to a JMX manager HTTP service using the HTTP protocol.	
--url	URL used to connect to a JMX manager's HTTP service.	
--username --user	The user name of the credential to use in authentication when connecting to the JMX manager. When specified, if the --password option is not also specified, gfsh will prompt for the password. This option cannot be used with --token.	
--password	The password portion of the credential to use in authentication when connecting to the JMX manager. This option cannot be used with --token.	

Name	Description	Default
--token	The OAuth token to use in authentication when connecting to the JMX manager. This token will be given to SecurityManager.authenticate on the JMX manager. This option cannot be used with --user or --password.	
--key-store	Java keystore file containing this application's certificate and private key. If the --key-store-password parameter is not specified, gfsh prompts the operator for the password.	
--key-store-password	Password to access the private key from the keystore file specified by --key-store.	
--trust-store	Java keystore file containing the collection of CA certificates trusted by this application. If the --trust-store-password parameter is not specified, gfsh prompts the operator for the password.	
--trust-store-password	Password to unlock the keystore file specified by --trust-store.	
--ciphers	SSL/TLS ciphers used when encrypting the connection. The default is "any".	
--protocols	SSL/TLS protocol versions to enable when encrypting the connection. The default is "any".	
--security-properties-file	The gfsecurity.properties file for configuring gfsh to connect to the Locator/Manager. The file 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 also reads gfsecurity.properties. SSL Options take precedence over values set in the properties file. If none are specified, defaults are used.	• If the parameter is not specified: false • If the parameter is specified without a value: true
--skip-ssl-validation	When SSL communication is enabled and this option is specified or assigned the value true, this gfsh client accepts any SSL certificate, allowing this gfsh client to authenticate any locator or server to which it is connecting. This option exists to facilitate testing, and it is not intended for production systems.	false

Table 1. Connect Parameters

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=GeodeStymon, port=1099] ..
Successfully connected to: [host=GeodeStymon, port=1099]
```

Example of connecting to a remote locator over HTTP:

```
gfsh>connect --url="http://myLocatorHost.example.com:8080/gemfire-mgmt/v1"
Successfully connected to: Geode Manager's HTTP service @
http://myLocatorHost.example.com:8080/gemfire-mgmt/v1
```

Error Messages:

```
"Locator could not find a JMX Manager";
"jmx password must be specified.";
"Could not connect to : {0}. {1}";
"Could not find a Geode jmx-manager service running at {0}.";
"--token requires a value, for example --token=foo";
"--token cannot be combined with --user or --password";
"Could not connect to Geode Locator service at {0}."
```

create

This topic describes the create command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to create async-event-queues, disk-stores, gateway receivers, gateway senders, indexes, and regions.

- create async-event-queue

  Creates an asynchronous event queue for batching events before they are delivered by a gateway sender.

- create defined indexes

  Creates all the defined indexes.

- create disk-store

  Defines a pool of one or more disk stores, which can be used by regions and client subscription queues, and gateway sender queues for WAN distribution.

- create gateway-receiver

  Creates a gateway receiver. 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 .

- create gateway-sender

  Creates a gateway sender on one or more members of a cluster.

- create index

  Create an index that can be used when executing queries.

- create initializer

  Create an initializer to run during server startup. Cluster configuration must be enabled to use this command.

- create jndi-binding

Create a JNDI binding that specifies resource attributes which describe a JDBC connection.

- create region

  Create a region with given path and configuration.

The order in which components are created matters. For example, the recommendation for WAN setup is:

1. Create/start WAN senders first
2. Create Regions
3. Create/start WAN receivers last

This assures that when WAN receivers are started, their associated regions are in place. Otherwise, the create region command may fail if events are received before the region exists. For more on this topic, see Configuring a Multi-site (WAN) System.

create async-event-queue

Creates an asynchronous event queue for batching events before they are delivered by a gateway sender.

> You can configure multiple regions to share the same AsyncEventQueue if they use the same AsyncEventListener. The maximum combined number of AsyncEventQueues and GatewaySenders per cluster is 128.

See Configuring Multi-Site (WAN) Event Queues.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
create async-event-queue --id=value --listener=value [--groups=value(,value)*]
    [--parallel(=value)?] [--enable-batch-conflation(=value)?] [--batch-size=value]
    [--batch-time-interval=value] [--persistent(=value)?] [--disk-store=value]
    [--disk-synchronous(=value)?] [--max-queue-memory=value]
    [--dispatcher-threads=value] [--order-policy=value]
    [--gateway-event-filter=value(,value)*]
    [--gateway-event-substitution-filter=value]
    [--listener-param=value(,value)*] [--forward-expiration-destroy(=value)?]
    [--pause-event-processing(=value)?]
```

Parameters, create async-event-queue:

Name	Description	Default Value
--id	*Required*. ID of the asynchronous event queue	
--groups	The queue is created on all members of the groups. If you do not specify a group, the queue is created on all members.	
--parallel	Specifies whether the queue is parallel.	false
--enable-batch-conflation	Enables batch conflation.	false
--batch-size	Maximum number of messages that a batch can contain.	100

Name	Description	Default Value
--batch-time-interval	Maximum amount of time, in ms, that can elapse before a batch is delivered, when no events are found in the queue to reach the batch-size.	5
--persistent	Boolean value that determines whether Tanzu GemFire persists this queue.	false If specified with out a value, default is true.
--disk-store	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, Tanzu GemFire uses the default disk store for overflow and queue persistence.	
--disk-synchronous	Specifies whether disk writes are synchronous.	true
--max-queue-memory	Maximum amount of memory in megabytes that the queue can consume before overflowing to disk.	100
--dispatcher-threads	Number of threads used for sending events.	5
--order-policy	Policy for dispatching events when --dispatcher-threads is greater than one. Possible values are THREAD, KEY, PARTITION.	KEY
--gateway-event-filter	List of fully qualified class names of GatewayEventFilters for this queue. These classes filter events before dispatching to remote servers.	
--gateway-event-substitution-filter	Fully-qualified class name of the GatewayEventSubstitutionFilter for this queue.	
--listener	*Required.* Fully-qualified class name of Async Event Listener for this queue	
--listener-param	Parameter name and value to be passed to the Async Event Listener class. Optionally, you can specify a value by following the parameter name with the # character and the value. For example: `--listener-param=myParam#24`	
--forward-expiration-destroy	Enables forwarding of expiration destroy operations to AsyncEventListener instances. If specified without a value, this parameter is set to "false".	false
--pause-event-processing	Specifies whether event dispatching from the queue to the listeners will be paused when the AsyncEventQueue is started. If specified without a value, this parameter is set to "true".	false

Example Commands:

```
create async-event-queue --id=myAEQ --listener=myApp.myListener
```

create defined indexes

Creates all the defined indexes.

See also define index and clear defined indexes.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
create defined indexes [--members=value(,value)*] [--groups=value(,value)*]
```

Parameters, create defined indexes:

Name	Description	Default
--members	Name/Id of the members on which index will be created.	
--groups	The index will be created on all the members in the member groups.	

Example Commands:

```
create defined indexes
```

Sample Output:

```
gfsh>create defined indexes
Indexes successfully created. Use list indexes to get details.
1. ubuntu(server1:17682)<v1>:27574
```

If index creation fails, you may receive an error message in gfsh similar to the following:

```
gfsh>create defined indexes
Exception : org.apache.geode.cache.query.RegionNotFoundException ,
Message : Region ' /r3' not found: from  /r3Occurred on following members
1. india(s1:17866)<v1>:27809
```

create disk-store

Defines a pool of one or more disk stores, which can be used by regions and client subscription queues, and gateway sender queues for WAN distribution.

See Disk Storage

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
create disk-store --name=value --dir=value(,value)* [--allow-force-compaction(=value)?]
[--auto-compact(=value)?] [--compaction-threshold=value] [--max-oplog-size=value]
[--queue-size=value] [--time-interval=value] [--write-buffer-size=value]
[--segments=value] [--groups=value(,value)*]
[--disk-usage-warning-percentage=value] [--disk-usage-critical-percentage=value]
```

Parameters, create disk-store:

Name	Description	Default Value
--name	*Required.* The name of this disk store.	

--dir	*Required.* One or more directory names where the disk store files are written. Optionally, directory names may be followed by # and the maximum number of megabytes that the disk store can use in the directory. For example:
	`--dir=/data/ds1`
	`--dir=/data/ds2#5000`
	If the specified directory does not exist, the command will create the directory for you.
		If the maximum directory size in megabytes is not specified, it will be set to 2147483647 (the value of Integer.MAX_VALUE)

--allow-force-compaction	Set to true to allow disk compaction to be forced on this disk store.	false
--auto-compact	Set to true to automatically compact the disk files.	true
--compaction-threshold	Percentage of non-garbage remaining, below which the disk store is eligible for compaction.	50
--max-oplog-size	Maximum size, in megabytes, for an oplog file. When the oplog file reaches this size, the file is rolled over to a new file.	1024
--queue-size	Maximum number of operations that can be asynchronously queued to be written to disk.	0
--time-interval	The number of milliseconds that can elapse before unwritten data is written to disk.	1000
--write-buffer-size	The size in bytes 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.	32768
--segments	The number of segments across which regions and partitioned region buckets will be spread.	Current number of available CPUs on the server.
--groups	The disk store is created on all members of the groups. If no group is specified, the disk store is created on all members.	
--disk-usage-warning-percentage	Disk usage above this threshold generates a warning message. For example, if the threshold is set to 90%, then on a 1 TB drive falling under 100 GB of free disk space generates the warning. Set to "0" (zero) to deactivate.	90
--disk-usage-critical-percentage	Disk usage above this threshold generates an error message and shuts down the member's cache. For example, if the threshold is set to 99%, then falling under 10 GB of free disk space on a 1 TB drive generates the error and shuts down the cache. Set to "0" (zero) to deactivate.	99

Example Commands:

```
create disk-store --name-store1 --dir=/data/ds1
```

Sample Output:

```
gfsh>create disk-store --name-store1 --dir=/data/ds1
Member  | Result
------- | -------
server1 | Success
```

create gateway-receiver

Creates gateway receivers. 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.

The create occurs on all servers, unless the --groups or --members option is specified.

If the gateway receiver creation succeeds on at least one member, this gfsh command exits with an exit code indicating success.

Outputs a tabular format status of each member's gateway receiver, independent of the success or failure of the creation.

See Gateway Receivers.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
create gateway-receiver [--groups=value(,value)*] [--members=value(,value)*]
  [--manual-start=(value)?] [--start-port=value] [--end-port=value] [--bind-address=value]
  [--maximum-time-between-pings=value] [--socket-buffer-size=value]
  [--hostname-for-senders=value]
  [--if-not-exists=(value)?]
```

Parameters, create gateway-receiver:

Name	Description	Default Value
--groups	Gateway receivers are created on the members of the groups.	
--members	Name of the members on which to create the gateway receiver. For backward compatibility, no gateway receiver configuration is persisted if this option is specified and cluster configuration is enabled.	
--manual-start	Boolean value that specifies whether you need to manually start the gateway receiver. When specified without providing a boolean value or when specified and set to "true", the gateway receiver must be started manually.	false
--start-port	Starting port number to use when specifying the range of possible port numbers this gateway receiver will use to connects to gateway senders in other sites. Tanzu 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 start-port and end-port values are inclusive. For example, if you specify start-port="50510" and end-port="50520", Tanzu GemFire chooses a port value from 50510 to 50520.	5000

Name	Description	Default Value
--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. Tanzu 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 end-port and start-port values are inclusive. For example, if you specify start-port="50510" and end-port="50520", Tanzu GemFire chooses a port value from 50510 to 50520.	5500
--bind-address	Network address for connections from gateway senders in other sites. Specify the address as a literal string value.	
--socket-buffer-size	An integer value that sets the buffer size (in bytes) of the socket connection for this gateway receiver. This value must be greater than zero; otherwise, creating the sender or receiver will fail. This value should match the socket-buffer-size setting of gateway senders that connect to this receiver.	524288
--gateway-transport-filter	(Deprecated) The fully qualified class name of the GatewayTransportFilter to be added to the Gateway receiver.	
--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
--hostname-for-senders	The host name or IP address told to gateway senders as the address for them to connect to. The locator informs gateway senders of this value.	
--if-not-exists	When specified without providing a boolean value or when specified and set to "true", gateway receivers will not be created if they already exist. Command output reports the status of each creation attempt.	false

Example Commands:

```
gfsh>create gateway-receiver --members=server1
```

Sample Output:

```
gfsh>create gateway-receiver --members=server1
Member  | Status
------- | ---------------------------------------------------------------------------
server1 | GatewayReceiver created on member "server1" and will listen on the port "0"
```

create gateway-sender

Creates a gateway sender on one or more members of a cluster.

See Gateway Senders.

> The gateway sender configuration for a specific sender id must be identical on each Tanzu GemFire member that hosts the gateway sender.

> You can configure multiple regions to share the same AsyncEventQueue if they use the same AsyncEventListener. The maximum combined number of AsyncEventQueues and GatewaySenders per cluster is 128.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
create gateway-sender --id=value --remote-distributed-system-id=value
  [--groups=value(,value)*] [--members=value(,value)*] [--parallel(=value)?]
  [--manual-start=value] [--socket-buffer-size=value] [--socket-read-timeout=value]
  [--enable-batch-conflation=value] [--batch-size=value] [--batch-time-interval=value]
  [--enable-persistence=value] [--disk-store-name=value] [--disk-synchronous=value]
  [--maximum-queue-memory=value] [--alert-threshold=value] [--dispatcher-threads=value]
  [--order-policy=value][--gateway-event-filter=value(,value)*]
```

Parameters, create gateway-sender:

Name	Description	Default
--id	*Required.* Unique identifier for the gateway sender, usually an identifier associated with a physical location.	
--remote-distributed-system-id	*Required.* ID of the remote cluster where this gateway sender sends events.	
--groups	Gateway senders are created on the members of the groups.	
--members	Name of the members on which to create the gateway sender.	
--parallel	When set to true, specifies a parallel Gateway Sender.	false
--enable-batch-conflation	Boolean value that determines whether Tanzu GemFire should conflate messages.	false
--manual-start	Deprecated. Boolean value that specifies whether you need to manually start the gateway sender. If you supply a null value, the default value of false is used, and the gateway sender starts automatically. *A manual start is likely to cause data loss, so manual start should never be used in a production system.*	false
--socket-buffer-size	Size of the socket buffer that sends messages to remote sites. This value must be greater than zero; otherwise, creating the sender or receiver will fail. This size should match the size of the socket-buffer-size attribute of remote gateway receivers that process region events.	524288
--socket-read-timeout	Amount of time in milliseconds that the gateway sender will wait to receive an acknowledgment from a remote site. By default this is set to 0, which means there is no timeout. If you do set this timeout, you must set it to a minimum of 30000 (milliseconds). Setting it to a lower number will generate an error message and reset the value to the default of 0.	0
--batch-size	Maximum number of messages that a batch can contain.	100
--batch-time-interval	Maximum amount of time, in ms, that can elapse before a batch is delivered, when no events are found in the queue to reach the batch-size.	1000
--enable-persistence	Boolean value that determines whether Tanzu GemFire persists the gateway queue.	false

Name	Description	Default
--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, Tanzu 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 Tanzu GemFire logs an alert.	0
--dispatcher-threads	Number of dispatcher threads that are used to process region events from a gateway sender queue or asynchronous event queue.	5
--order-policy	When the dispatcher-threads attribute is greater than 1, order-policy configures the way in which multiple dispatcher threads process region events from a serial gateway queue or serial asynchronous event queue. This attribute can have one of the following values: 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 Tanzu 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. You cannot configure the order-policy for a parallel event queue, because parallel queues cannot preserve event ordering for regions. Only the ordering of events for a given partition (or in a given queue of a distributed region) can be preserved.	key
--gateway-event-filter	A list of fully-qualified class names of GatewayEventFilters (separated by commas) to be associated with the GatewaySender. This serves as a callback for users to filter out events before dispatching to a remote cluster. For example: `gateway-event-filter=com.user.filters.MyFilter1,com.user.filters.MyFilters2`	
--gateway-transport-filter	(Deprecated) The fully-qualified class name of the GatewayTransportFilter to be added to the GatewaySender.	

Name	Description	Default
--enforce-threads-connect-same-receiver	This parameter applies only to serial gateway senders. If true, receiver member id is checked by all dispatcher threads when the connection is established to ensure they connect to the same receiver. Instead of starting all dispatcher threads in parallel, one thread is started first, and after that the rest are started in parallel.	false

Example Commands:

```
gfsh>create gateway-sender --remote-distributed-system-id="2" --id="sender2"
```

Sample Output:

```
gfsh>create gateway-sender --remote-distributed-system-id="2" --id="sender2"
Member  | Status
------- | ---------------------------------------------
server1 | GatewaySender "sender2" created on "server1"
```

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.

See Working with Indexes.

Syntax:

```
create index --name=value --expression=value --region=value
[--members=value(,value)*] [--type=value] [--groups=value(,value)*]
```

Parameters, create index:

Name	Description	Default
--name	*Required.* Name of 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.	
--members	Name/ID of the members on which index will be created.	
--type	Type of the index. Valid values are: range and key. (A third type, hash, is still recognized but hash indexes are deprecated.)	range
--groups	The index will be created on all the members in the groups.	

Example Commands:

```
create index --name=myKeyIndex --expression=region1.Id --region=region1 --type=key
```

Sample Output:

```
gfsh>create index --name=myKeyIdex --expression=region1.Id --region=region1 --type=key
Index successfully created with following details
Name       : myKeyIdex
Expression : region1.Id
RegionPath : /region1
Members which contain the index
1. ubuntu(server1:17682)<v1>:27574

gfsh>create index --name=myIndex2 --expression=exp2 --region=/exampleRegion
Failed to create index "myIndex2" due to following reasons
Index "myIndex2" already exists.  Create failed due to duplicate name.
Occurred on following members
1. ubuntu(server1:17682)<v1>:27574
```

create initializer

Create an initializer to be run during server start up. Cluster configuration must be enabled to use this command. Initializer classes must implement either the Declarable or CacheCallback interface. The CacheCallback interface includes an additional close method for shutting down and cleaning up resources started by the initializer. Initializers are registered with a group. Each group can only have one initializer. If no group is defined, the initializer will apply to the entire cluster.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
create initializer --initializer=classname{properties} [--groups=value(,value)*]
[--run-on-create(=value)?]
```

Parameters, create initializer:

Name	Description	Default
--initializer	*Required.* Fully qualified class name of the initializer. Must implement Declarable or CacheCallback. Optional JSON properties may be appended to class name.	
--groups	The initializer is configured for the provided groups. If no groups are specified, the initializer will apply to the entire cluster.	
--run-on-create	If true, the initializer will be created and run immediately on members in the specified groups.	false

Example Commands:

```
create initializer --initializer=com.example.MyInitializer --groups=group1 --run-on-create

create initializer --initializer=com.example.MyInitializer{"my-property1":"my-property1-value"}
```

create jndi-binding

Create a JNDI binding that specifies resource attributes which describe a JDBC connection.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
create jndi-binding --name=value --url=value
 [--jdbc-driver-class=value] [--type=value] [--blocking-timeout-seconds=value]
 [--conn-pooled-datasource-class=value] [--idle-timeout-seconds=value]
 [--init-pool-size=value] [--login-timeout-seconds=value]
 [--managed-conn-factory-class=value] [--max-pool-size=value] [--password=value]
 [--transaction-type=value] [--username=value] [--xa-datasource-class=value]
 [--if-not-exists(=value)?] [--datasource-config-properties=value(,value)*]
```

Parameters, create jndi-binding:

Name	Description	Default
--name	*Required.* Name of the binding to create.	
--url or --connection-url	*Required.* the JDBC driver connection URL string. For example, jdbc:hsqldb:hsql://localhost:1701.	
--jdbc-driver-class	The fully qualified name of the JDBC driver class.	
--type	Type of the XA datasource. One of: MANAGED, SIMPLE, POOLED, or XAPOOLED. If --type=POOLED and a --conn-pooled-datasource-class option is not specified, a pool will be created using Hikari. For more information about Hikari, see https://brettwooldridge.github.io/HikariCP.	SIMPLE
--blocking-timeout-seconds	Specifies the maximum time, in seconds, to block while waiting for a connection before throwing an exception.	
--conn-pooled-datasource-class	The fully qualified name of the connection pool implementation that holds XA datasource connections. If --type=POOLED, then this class must implement org.apache.geode.datasource.PooledDataSourceFactory.	
--idle-timeout-seconds	Specifies the time, in seconds, that a connection may be idle before being closed.	
--init-pool-size	Specifies the initial number of connections the pool should hold.	
--login-timeout-seconds	The quantity of seconds after which the client thread will be disconnected due to inactivity.	
--managed-conn-factory-class	The fully qualified name of the connection factory implementation.	
--max-pool-size	The maximum number of connections that may be created in a pool.	
--password	The default password used when creating a new connection.	
--transaction-type	Type of the transaction. One of XATransaction, NoTransaction, or LocalTransaction.	
--username	Specifies the user name to be used when creating a new connection. When specified, if the --password option is not also specified, gfsh will prompt for the password.	
--xa-datasource-class	The fully qualified name of the javax.sql.XADataSource implementation class.	

Name	Description	Default
--if-not-exists	When true, a duplicate jndi binding will not be created if one with the same name already exists. When false, an attempt to create a duplicate jndi binding results in an error. The option is set to true if the option is specified without a value.	false
--datasource-config-properties	Properties for the custom XADataSource driver. Append a JSON string containing a (name, type, value) tuple to set any property. If --type=POOLED, the properties will configure the database data source. If --type=POOLED and the value of a name within the tuple begins with the string “pool.”, then the properties will configure the pool data source. For example: --datasource-config-properties={'name':'name1','type':'type1','value':'value1'},{'name':'pool.name2','type':'type2','value':'value2'}	

Example Commands:

```
gfsh>create jndi-binding --name=jndi1 --type=SIMPLE \
  --jdbc-driver-class=org.apache.derby.jdbc.EmbeddedDriver \
  --url="jdbc:derby:newDB;create=true"
```

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.

See Region Data Storage and Distribution.

See Specifying JSON within Command-Line Options for syntax details.

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]
   [--groups=value(,value)*] [--if-not-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]
   [--entry-time-to-live-reset-policy=value]
   [--entry-idle-time-custom-expiry=value] [--entry-time-to-live-custom-expiry=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] [--compressor=value] [--off-heap(=value)?]
   [--partition-listener=value(,value)*] [--partition-resolver=value]
   [--eviction-entry-count=value] [--scope=value]
   [--eviction-max-memory=value] [--eviction-action=value]
   [--eviction-object-sizer=value]
```

Parameters, create region:

Name	Description	Default
--name	*Required.* Name/Path of the region to be created.	
--type	*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 --type parameter and then select the TAB key to display a full list.	
--template-region	*Required* (if type is not specified.) Name/Path of the region whose attributes should be duplicated when creating this region.	
--groups	Groups of members on which the region will be created.	
--if-not-exists	A new region will not be created if a region with the same name already exists. By default, an attempt to create a duplicate region is reported as an error. If this option is specified without a value or is specified with a value of true, then gfsh displays a "Skipping..." acknowledgement, but does not throw an error.	false
--key-constraint	Fully qualified class name of the objects allowed as region keys. Ensures that keys for region entries are all of the same class.	
--value-constraint	Fully qualified class name of the objects allowed as region values. If not specified, then region values can be of any class.	
--enable-statistics	Whether to gather statistics for the region. Must be true to use expiration on the region.	
--entry-idle-time-expiration	How long, in seconds, the region's entries can remain in the cache without being accessed.	no expiration
--entry-idle-time-expiration-action	Action to be taken on an entry that has exceeded the idle expiration. Valid expiration actions include destroy, local-destroy, invalidate (default), local-invalidate.	
--entry-time-to-live-expiration	How long, in seconds, the region's entries can remain in the cache without being accessed or updated. The default is no expiration of this type.	no expiration
--entry-time-to-live-expiration-action	Action to be taken on an entry that has exceeded the TTL expiration. Valid expiration actions include destroy, local-destroy, invalidate (default), local-invalidate.	
--entry-time-to-live-reset-policy	Determines if the entry's timeout should be relative to its creation time or the last modified time. Valid options are never and last-modified.	last-modified
--entry-idle-time-custom-expiry	The name of a class implementing CustomExpiry for entry idle time. Append a JSON string for initialization properties.	
--entry-time-to-live-custom-expiry	The name of a class implementing CustomExpiry for entry time to live. Append a JSON string for initialization properties.	
--region-idle-time-expiration	Deprecated.How long, in seconds, the region can remain in the cache without its entries being accessed. The default is no expiration of this type.	

Name	Description	Default
--region-idle-time-expiration-action	Deprecated.Action to be taken on a region that has exceeded the idle expiration. Valid expiration actions include destroy, local-destroy, invalidate (default), local-invalidate. The destroy and local-destroy actions destroy the region. The invalidate and local-invalidate actions leave the region in place, but invalidate all of its entries.	
--region-time-to-live-expiration	Deprecated.How long, in seconds, the region can remain in the cache without its entries being accessed or updated. The default is no expiration of this type.	no expiration
--region-time-to-live-expiration-action	Deprecated.Action to be taken on a region that has exceeded the TTL expiration. Valid expiration actions include destroy, local-destroy, invalidate (default), local-invalidate. The destroy and local-destroy actions destroy the region. The invalidate and local-invalidate actions leave the region in place, but invalidate all of its entries.	
--disk-store	Disk Store to be used by this region. The list disk-stores 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. A fully qualified class name may be appended with a JSON specification that will be parsed to become the fields of the parameter to the init() method for a class that implements the Declarable interface.	
--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. A fully qualified class name may be appended with a JSON specification that will be parsed to become the fields of the parameter to the initialize() method for a class that implements the Declarable interface.	
--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. A fully qualified class name may be appended with a JSON specification that will be parsed to become the fields of the parameter to the init() method for a class that implements the Declarable interface.	
--async-event-queue-id	IDs of the Async Event Queues that will be used for write-behind operations.	

Name	Description	Default
--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 after a member crashes before restoring this region's redundancy on the remaining members. The default value (-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 assuming their share of cluster-level redundancy. This allows time for multiple regions to start before the redundancy workload is parceled out to the new members. A value of -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.	113
--compressor	Java class name that implements compression for the region. You can write a custom compressor that implements org.apache.geode.compression.Compressor or you can specify the Snappy compressor (org.apache.geode.compression.SnappyCompressor), which is bundled with Tanzu GemFire. See Region Compression.	no compression
--off-heap	Specifies whether the region values are stored in heap memory or off-heap memory. When true, region values are in off-heap memory. If the parameter is specified without a value, the value of true is used.	false
--partition-listener	Specifies fully-qualified class names of one or more custom partition listeners.	
--partition-resolver	Specifies the full path to a custom partition resolver. Specify org.apache.geode.cache.util.StringPrefixPartitionResolver to use the included string prefix PartitionResolver.	
--eviction-entry-count	Enables eviction, where the eviction policy is based on the number of entries in the region.	

Name	Description	Default
--eviction-max-memory	Enables eviction, where the eviction policy is based on the amount of memory consumed by the region, specified in megabytes.	
--eviction-action	Action to take when the eviction threshold is reached. local-destroy: Entry is destroyed locally. Use with caution because this can lead to inconsistencies. overflow-to-disk: Entry is overflowed to disk. For partitioned regions, this provides the most reliable read behavior across the region.	
--eviction-object-sizer	Specifies your implementation of the ObjectSizer interface to measure the size of objects in the region. The sizer applies only to heap and memory based eviction.	
--scope	Specifies the scope of the Replicated region. This field can be only used if the --type parameter is set with a replicated region type. This parameter is invalid for all other region types. If this parameter is not set and --type is set to a replicated region, the default scope DISTRIBUTED_ACK is set.	

Example Commands:

```
create region --name=region1 --type=REPLICATE_PERSISTENT \
--cache-writer=org.apache.geode.examples.MyCacheWriter \
--group=Group1 --disk-store=DiskStore1

create region --name=region12 --template-region=/region1

create region --name=region2 --type=REPLICATE \
--cache-listener=org.apache.geode.examples.MyCacheListener1,\
org.apache.geode.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=org.apache.geode.examples.MyCacheListener3 \
--group=Group2

create region --name=region4 --type=REPLICATE_PROXY \
--cache-listener=org.apache.geode.examples.MyCacheListener1 --group=Group1,Group2

create region --name=myRegion --type=REPLICATE --eviction-max-memory=100 \
--eviction-action=overflow-to-disk --eviction-object-sizer=my.company.geode.MySizer

create region --name=r1 --type=PARTITION \
--cache-loader=org.example.myLoader{'URL':'jdbc:cloudscape:rmi:MyData'}
```

Sample Output:

```
gfsh>create region --name=myRegion --type=LOCAL
Member  | Status
------- | --------------------------------------
server1 | Region "/myRegion" created on "server1"
```

debug

This topic describes the debug command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to activate or deactivate debugging output in gfsh.

Availability: Online or offline.

Syntax:

```
debug --state=value
```

Name	Description	Default Value
--state	Whether to turn debugging ON or OFF. Valid options are: ON, OFF (Case insensitive)	OFF

Table 1. Debug Parameters

Example Commands:

```
debug --state=off
debug --state=on
```

Sample Output:

```
gfsh>debug --state=on
Debug is on
```

define index

This topic describes the define command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to define an index that can be used when executing queries. This allows you to execute a single command to create multiple indexes concurrently using create defined indexes.

Availability: Online or offline.

Syntax:

```
define index --name=value --expression=value --region=value [--type=value]
```

Name	Description	Default Value
--name	*Required.* Name of the index to define.	
--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.	
--type	Type of the index. Valid values are: range, key and hash.	range

Table 1. Define Index Parameters

Example Commands:

```
gfsh> define index --name=myIndex1 --expression=exp1 --region=/exampleRegion

gfsh> define index --name=myIndex2 --expression="c.exp2" --region="/exampleRegion e, e.collection1 c"

gfsh> define index --name=myIndex3 --expression=exp3 --region=/exampleRegion --type=hash

//then to create the indexes, execute:

gfsh> create defined indexes
```

Sample Output:

```
gfsh>define index --name=myIndex1 --expression=exp1 --region=/exampleRegion
Index successfully defined with following details
Name       : myIndex1
Expression : exp1
RegionPath : /exampleRegion
```

deploy

This topic describes the deploy command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to deploy JAR-packaged applications to a member or members.

> When deploying a JAR, the name of the JAR cannot contain any of the following characters: *, !, #, @, ?.

Only one of either --jars or --dir may be specified.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
deploy [--groups=value(,value)*] [--jars=value(,value)*] [--dir=value]
  [--dependencies=value]
```

Name	Description
--groups	Groups to which the specified JARs will be deployed. If this option is not specified, the deployment will occur on all members.
--jars	Paths of the JARs to deploy.
--dir	Directory from which to deploy the JARs.
--dependencies	Names of other deployments on which this deployment should depend. This deployment will be able to access classes defined in the listed deployments. This option is not valid when classloader isolation is disabled.

Table 1. Deploy Parameters

Example Commands:

```
deploy --jars=group1_functions.jar --groups=Group1

deploy --dir=libs/group1-libs --groups=Group2
```

Sample Output:

```
gfsh> deploy --jars=group1_functions.jar --groups=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 --groups=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

This topic describes the describe command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to display details of a member's configuration, shell connection, disk-stores, members, or regions.

- describe async-event-queue

  Displays the details of the asynchronous event queue created in the cluster members.

- describe client

  Displays details about a specified client.

- describe config

  Display the configuration of a member.

- describe connection

  Display connection information details.

- describe disk-store

  Display information about a member's disk store.

- describe gateway-receiver

  Display details about a gateway receiver created in the cluster members.

- describe gateway-sender

  Display details about a gateway sender created in the cluster members.

- describe jndi-binding

  Display information about the configuration of a JNDI binding.

- describe member

  Display details of a member with given name or ID.

- describe offline-disk-store

Display information about an offline member's disk store.

- describe query

  Display additional details about an executing query.

- describe query-service

  Display information about the configuration of the query service.

- describe region

  Display the attributes and key information of a region.

- describe wan-copy region

  Display detailed information about a running wan-copy region command.

---

describe async-event-queue

Displays details about an asynchronous event queue created in the cluster members.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
describe async-event-queue --id=value [--members=value(,value)*] [--groups=value(,value)*]
```

Parameters, describe async-event-queue:

Name	Description
--id	*Required.* ID of the asynchronous event queue whose details are to be described.
--member --members	Optional. [Optional] Name and ID of member or members whose asynchronous event queue details are to be described.
--group --groups	Optional. Group or groups of members whose asynchronous event queue details are to be described.

Example Commands:

```
describe async-event-queue --id=queue1
describe async-event-queue --id=queue1 --members=server1
describe async-event-queue --id=queue1 --groups=group1
```

Output:

Property	Description
Asynchronous event queue ID	The unique identifier associated with the asynchronous event queue.
Hosting members	Members in the cluster which contain the asynchronous event queue.
AEQ Listener class name	Fully-qualified class name of the listeners.
Batch conflation enabled	Whether or not batch conflation is enabled. • true: Events in queue are conflated. • false: Events in the queue are not conflated.
Batch size	Maximum number of messages in a batch that are sent for processing.

Property	Description
Batch time interval (ms)	Time, in milliseconds, that a member waits before sending a partially-filled batch for processing if there are no incoming events.
Disk synchronous	Whether or not writes to the disk are synchronous. • true: Writes to the disk are synchronous. • false: Writes to the disk are not synchronous.
Disk Store Name	Name of the disk store associated with the asynchronous event queue.
Forward expiration destroy	Whether or not destroy operations from expiration are forwarded to the queue. • true: destroy operations from expiration are forwarded to the queue. • false: destroy operations from expirations are not forwarded to the queue.
Gateway event filter class name	Fully-qualified class name of the gateway event filters associated with the queue.
Gateway event substitution filter	Fully-qualified class name of the gateway event substitution filter associated with the queue.
Maximum queue memory (MB)	Maximum memory, in megabytes, for the queue, after which entries overflow to the disk.
Number of dispatcher threads	Number of dispatcher threads working for the queue.
Order policy	Policy followed while dispatching events to the listeners. • KEY • THREAD • PARTITION
Parallel	Whether the queue is parallel or serial. • true: Parallel queue. • false: Serial queue.
Persistence enabled	Whether or not the queue is configured to be persistent. • true: Queue is configured to be persistent. • false: Queue is not configured to be persistent.
Regions associated with the async event queue	Regions linked to the asynchronous event queue.
Status	Whether or not the queue is ready for dispatching events to the listener. • READY FOR DISPATCHING: Queue is ready to dispatch events to the listener. • PAUSED: Queue will not dispatch the events to the listener.
Is primary	Which member is the primary for the asynchronous event queue.

Sample Output:

```
gfsh>describe async-event-queue --id=queue1

Asynchronous event queue ID : queue1
Hosting members             : server-1,server-3

Properties shared by members:

                 Property                  | Value
---------------------------------------------- | -------------------------------------
AEQ Listener class name                      | MyAsyncEventListener
Batch conflation enabled                     | false
Batch size                                   | 100
Batch time interval (ms)                     | 5
Disk synchronous                             | true
Diskstore name                               | DEFAULT
Forward expiration destroy                   | false
Gateway event filter class name              | TestGatewayEventFilter
Gateway event substitution filter            | StringGatewayEventSubstitutionFilter
Maximum queue memory (MB)                    | 100
Number of dispatcher threads                 | 1
Order policy                                 | KEY
Parallel                                     | false
Persistence enabled                          | false
Regions associated with the async event queue | /region1
                                             | /region2
                                             | /region3
Status                                       | READY FOR DISPATCHING

----------------------------------------------------------------
Member specific properties:

 Member  |  Property  | Value
-------- | ---------- | -----
server-1 | Is primary | true
server-3 | Is primary | false

----------------------------------------------------------------
Members that did not return any information about queue id:queue1
server-2
----------------------------------------------------------------
```

---

describe client

Displays details about a specified client.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
describe client --clientID=value
```

Parameters, describe client:

Name	Description
--clientID	*Required.* ID of the client. To find a client ID, you can use the list clients command to display a list of connected clients and their IDs.

Example Command:

```
describe client --clientID=192.0.2.0(4987:loner):58922:7b3398cf
```

Sample Output:

```
gfsh>describe client --clientID=192.0.2.0(4987:loner):58922:7b3398cf

----------------------------------------------------------------
Primary Servers              : 192.0.2.0(server1:5764)<v1>:15189
Secondary Servers            : 192.0.2.0(server2:5891)<v2>:39082
CPU                          : 0
Number of Cache Listener Calls : 0
Number of Gets               : 0
Number of Misses             : 0
Number of Puts               : 0
Number of Threads            : 0
Process CPU Time (nanoseconds) : 0
Queue size                   : 1
UP Time (seconds)            : 67
Is Durable                   : No
```

---

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)?]
```

Parameters, describe config:

Name	Description	Default Value
--member	Name or ID of a member whose configuration is to be shown.	
--hide-defaults	Whether or not to hide configuration information for properties with the default value.	true

Example Command:

```
describe config --member=Member1;
```

Sample Output:

```
gfsh>describe config --member=server1

Configuration of member : "server1"


JVM command line arguments
------------------------------------
-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 Command:

```
describe connection
```

Sample Output:

```
gfsh>describe connection

Connection Endpoints
--------------------
GemFireUser[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
```

Parameters, describe disk-store:

Name	Description
--member	*Required*. Name or ID of the member with the disk store to be described.
--name	*Required*. Name of the disk store to be described.

Example Command:

```
describe disk-store --member=server1 --name=DiskStore1
```

Sample Output:

```
gfsh>describe disk-store --name=disk1 --member=server1

Disk Store ID                   : a531bc7b-5188-4510-85d7-de7de30c6671
Disk Store Name                 : disk1
Segments                        : 42
Member ID                       : ubuntu(server1:7467)<v1>:35249
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
Disk Usage Warning Percentage   : 90
Disk Usage Critical Percentage  : 99
PDX Serialization Meta-Data Stored : No

      Disk Directory          | Size
------------------------------- | ----------
/home/user/server1/DiskStore1   | 2147483647
```

> Segments will not be displayed if the disk store is a legacy disk store.

---

describe gateway-receiver

Display details about a gateway receiver created in the cluster members.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
describe gateway-receiver [--members=values(,values)*] [--groups=value(,value)*]
```

Parameters, describe gateway-receiver:

Name	Description
--member --members	Optional. Member or members with the gateway receiver to describe.
--group --groups	Optional. Group or groups of members for which the gateway receivers will be described.

Example Commands:

```
describe gateway-receiver
describe gateway-receiver --member=server-2
describe gateway-receiver --group=group1
```

Output:

Property	Description
Hosting Members	Members hosting the gateway receiver.
Gateway transport filter	(Deprecated) All gateway transport filters associated with the gateway receiver.
End port	End limit of port range used by the receiver.
Hostname for senders	Hostname associated with receiver which the senders use to connect.
Maximum time between pings (ms)	Time, in millisecond, between pings to connected sites.
Socket buffer size (bytes)	Buffer size, in bytes, of the socket connection associated with the receiver.
Start port	Starting port number for the range used by the receiver.
Is Running	Whether or not the gateway receiver is in a running state.
Port	Port assigned to the receiver.
Sender count	Number of senders connected to the receiver.
Senders connected	Hosts and ports of all senders connected to the receiver.

Sample Output:

```
Hosting members : server-4,server-3

Properties shared by members:

          Property             | Values
------------------------------- | -------
End port                       | 29999
Gateway transport filter       | Not Available
Hostname for senders           |
Is Running                     | true
Maximum time between pings (ms) | 60000
Socket buffer size (bytes)     | 524288
Start port                     | 20001

-----------------------------------------------------------------
Member specific properties:

 Member  |     Property      | Values
-------- | ----------------- | --------------------------------------
server-4 | Port              | 27144
         | Sender count      | 3
         | Senders connected | 192.168.50.8(server-6:75815)<v2>:55056
         |                   | 192.168.50.8(server-6:75815)<v2>:55056
         |                   | 192.168.50.8(server-6:75815)<v2>:55056
server-3 | Port              | 22079
         | Sender count      | 12
         | Senders connected | 192.168.50.8(server-5:75814)<v1>:54593
         |                   | 192.168.50.8(server-5:75814)<v1>:54593
         |                   | 192.168.50.8(server-5:75814)<v1>:54593
         |                   | 192.168.50.8(server-5:75814)<v1>:54593
         |                   | 192.168.50.8(server-5:75814)<v1>:54593
         |                   | 192.168.50.8(server-6:75815)<v2>:55056
         |                   | 192.168.50.8(server-6:75815)<v2>:55056
         |                   | 192.168.50.8(server-6:75815)<v2>:55056
         |                   | 192.168.50.8(server-6:75815)<v2>:55056
         |                   | 192.168.50.8(server-6:75815)<v2>:55056
         |                   | 192.168.50.8(server-6:75815)<v2>:55056
         |                   | 192.168.50.8(server-6:75815)<v2>:55056
         |                   | 192.168.50.8(server-6:75815)<v2>:55056
```

describe gateway-sender

Display details about a gateway sender created in the cluster members.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
describe gateway-sender --id=value [--members=value(,value)*] [--groups=value(,value)*]
```

Parameters, describe gateway-sender:

Name	Description
--id	*Required.* ID of the gateway sender.
--member --members	Optional. Name of ID of the member or members with the gateway sender to describe.
--group --groups	Optional. Group or groups of members for which the gateway sender will be described.

Example Commands:

```
describe gateway-sender --id=In_Serial
describe gateway-sender --id=In_Serial --group=group1
describe gateway-sender --id=In_Serial --groups=group1,group2
describe gateway-sender --id=In_Serial --member=member1
describe gateway-sender --id=In_Serial --members=member1,member2
```

Output:

Property	Description
Gateway Sender ID	ID of the gateway sender.
Distributed System ID	ID of the member's distributed system where the gateway sender is running.
Remote Distributed System ID	Distributed system ID of the remote site to which the gateway sender is connected.
Hosting Members	Members that host the gateway sender.
Type	Type of gateway sender. Possible values are Serial and Parallel.
Batch Size	Maximum batch size that the gateway sender sends to the remote site.
Batch Time Interval (ms)	Maximum batch time interval, in milliseconds, that the gateway sender waits before attempting to send a batch of queued objects to the remote gateway receiver.
Persist Gateway Queue	Whether or not persistence is enabled for the gateway sender.
Disk Store	Name of the DiskStore used by the gateway sender when persisting the gateway sender queue's data.
Disk Synchronous	Whether or not the gateway sender is configured to use synchronous Disk Store writes.
Max Queue Memory (MB)	Maximum size, in megabytes, that the gateway sender's queue may take in heap memory before overflowing to disk.
Alert Threshold (ms)	Time, in milliseconds, that an object can be in the queue to be replicated before the gateway sender logs an alert.
Dispatcher Threads	Number of dispatcher threads that the gateway sender uses to dispatch the queued events.
Order Policy	Order policy followed while dispatching the events to a remote distributed system. Order policy is set only when the dispatcher threads are greater than one. Possible values: • THREAD - Indicates that events are parallelized based on the event's originating member and thread. • KEY - Indicates that events are parallelized based on the event's key. • PARTITION - In the case of a partitioned region event, indicates that events are parallelized based on the event's partition. In the case of a replicated region event, indicates that events are parallelized based on the event's key.
Enforce Threads Connect Same Receiver	Whether or not the sender threads must verify the receiver member ID to verify if they are connected to the same server.
Group Transaction Events	Whether or not the gateway sender is configured to send all events of a transaction in the same batch.
Gateway Event Filters	Fully-qualified class names of GatewayEventFilters associated with the gateway sender. This serves as a callback for users to filter out events before dispatching to a remote distributed system.

Property	Description
Gateway Transport Filters	(Deprecated) Fully-qualified class names of GatewayTransportFilters associated with the gateway sender.
Enable Batch Conflation	Whether or not the gateway sender is configured to use conflate entries in each batch.
Socket Buffer Size (bytes)	Buffer size, in bytes, of the socket connection between the gateway sender and its receiving gateway receiver.
Socket Read Timeout (ms)	Time, in milliseconds, that a socket read between a sending gateway sender and its receiving gateway receiver may block.
Regions	Region names that are configured with the gateway sender.

Sample Output:

```
Gateway Sender ID           : gatewaySender1
Distributed System ID       : 1
Remote Distributed System ID : 2
Hosting Members             : server-2, server-3, server-4, server-5

Properties Shared By Hosting Members

             Property                | Value
------------------------------------- | -------
Type                                 | Serial
Batch Time Interval (ms)             | 1000
Persist Gateway Queue                | false
Disk Store                           | DEFAULT
Disk Synchronous                     | true
Max Queue Memory (MB)                | 100
Dispatcher Threads                   | 5
Order Policy                         | KEY
Enforce Threads Connect Same Receiver | false
Enable Batch Conflation              | false
Socket Buffer Size (bytes)           | 524288
Socket Read Timeout                  | 0
Regions                              | Not Available
Gateway Transport Filters            | Not Available

Member-Specific Properties

 Member  |         Property         | Value
-------- | ------------------------ | ------------------------------------------------------------------------
server-2 | Batch Size               | 100
         | Alert Threshold (ms)     | 0
         | Group Transaction Events | false
         | Gateway Event Filters    | org.apache.geode.internal.cache.wan.wancommand.MyGatewayEventFilter1
         |                          | org.apache.geode.internal.cache.wan.wancommand.MyGatewayEventFilter2
server-3 | Batch Size               | 100
         | Alert Threshold (ms)     | 0
         | Group Transaction Events | false
server-4 | Batch Size               | 300
         | Alert Threshold (ms)     | 10
         | Group Transaction Events | false
server-5 | Batch Size               | 100
         | Alert Threshold (ms)     | 5
         | Group Transaction Events | true
```

describe jndi-binding

Print the configuration information that describes a JDBC connection.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
describe jndi-binding --name=value
```

Parameters, describe jndi-binding:

Name	Description
--name	*Required.* Name of the JNDI binding to be described.

Example Command:

```
describe jndi-binding --name=jndi1
```

Sample Output:

```
gfsh>describe jndi-binding --name=jndi1

    Property      | Value
----------------- | ------------------------------------
type              | SimpleDataSource
jndi-name         | jndi1
jdbc-driver-class | org.apache.derby.jdbc.EmbeddedDriver
user-name         |
connection-url    | jdbc:derby:newDB
```

describe member

Display details of a member with given name or ID.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
describe member --name=value
describe member --name=value [--verbose]
describe member --name=value [--configuration] [--extension][--membership-view][--status][--startup-lifecycle]
```

Parameters, describe member:

Name	Description
--name	*Required.* Display information about a member.
--verbose	Optional. Displays additional information about a member. This includes sections that displays the membership view, configuration, extensions, status, and the startup lifecycle progress of the member.
--configuration	Optional. Displays only the properties and system configurations of the member.
--extension	Optional. Displays only the extension services loaded on the member.
--membership-view	Optional. Displays only the membership view of the cluster as seen by the member.
--status	Optional. Displays only the current status of the member, including alerts, resource usage, and uptime.

Name	Description
--startup-lifecycle	Optional. Displays only the initialization status of the member's components during and after startup.

Example Command:

```
describe member --name=server1
```

Sample Output:

```
gfsh>describe member --name=server1

Name : server1
Id : GemFireUser(server1:240)<v1>:64871
Host : 192.0.2.0
Regions : region4
region5
region3
region2
region1
PID : 240
Groups :
Used Heap : 5M
Max Heap : 123M
Working Dir : c:\VMwareGemFire100\Latest\server1
Log file : C:\VMwareGemFire100\Latest\server1\vf.gf.server.log
Locators : localhost[10334]

Server Bind : localhost Server
Port : 40404
Running : true
Client Connections : 0
```

Example Command:

```
describe member --name=server1 --status
```

Sample Output:

```
gfsh>describe member --name=server1 --status

Name : server1
Id : GemFireUser(server1:240)<v1>:64871
Host : 192.0.2.0
Regions : region4
region5
region3
region2
region1
PID : 240
Groups :
Used Heap : 5M
Max Heap : 123M
Working Dir : c:\VMwareGemFire100\Latest\server1
Log file : C:\VMwareGemFire100\Latest\server1\vf.gf.server.log
Locators : localhost[10334]

Server Bind : localhost Server
Port : 40404
Running : true
Client Connections : 0
------------------------------
Status
------------------------------
Alerts :
  Statistics Sampler : No alert
  Memory Monitor : Low memory alert
  Eviction Status : Member: 172.20.10.5(server-2:48033)<v2>:46846 above heap eviction threshold
Uptime : 18 seconds
CPU use : 0%
Initial Heap : 512M
Used Heap : 146M
Max Heap : 491M
JVM Pauses : 0
```

Example Command:

```
describe member --name=server2 --verbose
```

Sample Output:

```
gfsh> describe member --name=server2 --verbose

Name            : server-2
Id              : 192.168.50.8(server-2:98952)<v2>:60157
Type            : Server
Host            : 192.168.50.8
Regions         :
Timezone        : America/Los_Angeles -07:00
Metrics URL     : Not Available
PID             : 98952
Groups          :
Redundancy-Zone : RACK1
Used Heap       : 145M
Max Heap        : 491M
Load Average1   : 11.21
Working Dir     : /test-worker-000001/dunit/vm2
Log file        : /test-worker-000001/dunit/vm2
Locators        : localhost[20004]

Cache Server Information
Server Bind             :
Server Port             : 20011
Running                 : true

Client Connections : 0

------------------------------
Startup Lifecycle
------------------------------
Membership Service Startup : Complete
Joined Cluster : Complete
Management Service Startup : Complete
Cache Server Startup : Complete
Gateway Receiver : Skipped
Gateway Senders : Skipped
Region Initialization : Skipped
Internal Region Init : Complete
Region Disk Recovery : Skipped

------------------------------
Status
------------------------------
Alerts :
  Statistics Sampler : No alert
  Memory Monitor : Low memory alert
  Eviction Status : Member: 172.20.10.5(server-2:48033)<v2>:46846 above heap eviction threshold
Uptime : 18 seconds
CPU use : 0%
Initial Heap : 512M
Used Heap : 146M
Max Heap : 491M
JVM Pauses : 0

------------------------------
Extensions
------------------------------
Test Extension
VMware Tanzu GemFire Prometheus Metrics

------------------------------
Membership View
------------------------------
View ID : 2
```

```
Members :
192.168.50.8(locator-0:98950:locator)<ec><v0>:50692
192.168.50.8(server-1:98951)<v1>:54927
192.168.50.8(server-2:98952)<v2>:60157

------------------------------
Configuration:
------------------------------
----------------------------------------
Test Extension
----------------------------------------
----------------------------------------
VMware Tanzu GemFire Prometheus Metrics
----------------------------------------
GemFire-Prometheus-Metrics-Version: 10.1.0-build.0
GemFire-Prometheus-Metrics-Source-Branch: feature/GEM-6216
GemFire-Prometheus-Metrics-Source-Date: 2023-09-18 10:47:20 -0700
GemFire-Prometheus-Metrics-Source-Revision: e726fc3ca559fc8d19da5ba550a6cf9b7fdc97ac
----------------------------------------
VMware GemFire
----------------------------------------
Build-Java-Vendor: BellSoft
GemFire-Source-Repository: feature/GEM-6216
GemFire-Source-Revision: e726fc3ca559fc8d19da5ba550a6cf9b7fdc97ac
Build-Id: nnag 0
GemFire-Source-Date: 2023-09-18 10:47:20 -0700
GemFire-Version: 10.1.0-build.0
Product-Name: VMware GemFire
Build-Platform: Mac OS X 10.16 x86_64
Build-Java-Version: 1.8.0_352

Running on: Running on: 192.168.50.8/192.168.50.8, 10 cpu(s), x86_64 Mac OS X 10.16

Communications version: 210
Process ID: 98952
User: nnag
Current dir: /Users/nnag/Development/gemfireDev/gemfire/gemfire/gemfire-gfsh/build/distributedTest/test-worker-000001/dunit/v
m2
Home dir: /Users/nnag
Command Line Parameters:
  -DDUnitLauncher.REMOTE_STUB_PORT=0
  -Dgemfire.DUnitLauncher.RMI_PORT=20001
  -Dgemfire.DUnitLauncher.VM_NUM=2
  -Dgemfire.DUnitLauncher.VM_VERSION=10240.0.0
  -DWORKSPACE_DIR=/Users/nnag/Development/gemfireDev/gemfire/gemfire/gemfire-gfsh/build/distributedTest/test-worker-00
0001/.
  -DAvailablePort.lowerBound=20001
  -DAvailablePort.upperBound=29999
  -DLocator.inhibitDMBanner=true
  -Xrunjdwp:transport=dt_socket,server=y,suspend=n
  -XX:+HeapDumpOnOutOfMemoryError
  -Xmx512m
  -Dgemfire.DEFAULT_MAX_OPLOG_SIZE=10
  -Dgemfire.disallowMcastDefaults=true
  -Dgemfire.use-ephemeral-ports=true
  -Dgemfire.validate-serializable-objects=true
  -ea
  -XX:MetaspaceSize=512m
  -XX:SoftRefLRUPolicyMSPerMB=1
  -DdummyArg=true
  -Djboss.modules.system.pkgs=javax.management,java.lang.management
  -Dboot.module.loader=com.vmware.gemfire.bootstrap.internal.GemFireModuleLoader
Class Path:
  /Users/nnag/Development/gemfireDev/gemfire/gemfire/gemfire-assembly/build/install/vmware-gemfire-distributed-testing/lib/ge
```

```
mfire-bootstrap-10.1.0-build.0.jar
Library Path:
  /Users/nnag/Library/Java/Extensions
  /Library/Java/Extensions
  /Network/Library/Java/Extensions
  /System/Library/Java/Extensions
  /usr/lib/java
  .
System Properties:
    AvailablePort.lowerBound = 20001
    AvailablePort.upperBound = 29999
    DUnitLauncher.REMOTE_STUB_PORT = 0
    Locator.inhibitDMBanner = true
    WORKSPACE_DIR = /Users/nnag/Development/gemfireDev/gemfire/gemfire/gemfire-gfsh/build/distributedTest/test-worker-00
0001/.
    awt.toolkit = sun.lwawt.macosx.LWCToolkit
    boot.module.loader = com.vmware.gemfire.bootstrap.internal.GemFireModuleLoader
    dummyArg = true
    file.encoding = UTF-8
    file.encoding.pkg = sun.io
    file.separator = /
    gemfire.DEFAULT_MAX_OPLOG_SIZE = 10
    gemfire.DUnitLauncher.LAUNCHED = true
    gemfire.DUnitLauncher.RMI_PORT = 20001
    gemfire.DUnitLauncher.VM_NUM = 2
    gemfire.DUnitLauncher.VM_VERSION = 10240.0.0
    gemfire.disallowMcastDefaults = true
    gemfire.free-off-heap-memory = true
    gemfire.use-ephemeral-ports = true
    gemfire.validate-serializable-objects = true
    gopherProxySet = false
    java.awt.graphicsenv = sun.awt.CGraphicsEnvironment
    java.awt.printerjob = sun.lwawt.macosx.CPrinterJob
    java.class.version = 52.0
    java.endorsed.dirs = /Library/Java/JavaVirtualMachines/liberica-jdk-8.jdk/Contents/Home/jre/lib/endorsed
    java.ext.dirs = /Users/nnag/Library/Java/Extensions:/Library/Java/JavaVirtualMachines/liberica-jdk-8.jdk/Contents/Home/jre/li
b/ext:/Library/Java/Extensions:/Network/Library/Java/Extensions:/System/Library/Java/Extensions:/usr/lib/java
    java.home = /Library/Java/JavaVirtualMachines/liberica-jdk-8.jdk/Contents/Home/jre
    java.io.tmpdir = /var/folders/5d/wdm0b23d4md1m02f50mp44cw0000gp/T/
    java.runtime.name = OpenJDK Runtime Environment
    java.runtime.version = 1.8.0_352-b08
    java.specification.maintenance.version = 4
    java.specification.name = Java Platform API Specification
    java.specification.vendor = Oracle Corporation
    java.specification.version = 1.8
    java.vendor = BellSoft
    java.vendor.url = https://bell-sw.com/
    java.vendor.url.bug = https://bell-sw.com/support
    java.version = 1.8.0_352
    java.vm.info = mixed mode
    java.vm.name = OpenJDK 64-Bit Server VM
    java.vm.specification.name = Java Virtual Machine Specification
    java.vm.specification.vendor = Oracle Corporation
    java.vm.specification.version = 1.8
    java.vm.vendor = BellSoft
    java.vm.version = 25.352-b08
    jboss.modules.system.pkgs = javax.management,java.lang.management
    line.separator =

    module.path = /Users/nnag/Development/gemfireDev/gemfire/gemfire/gemfire-assembly/build/install/vmware-gemfire-distribut
ed-testing/moduleDescriptors/gemfire-gfsh:/Users/nnag/Development/gemfireDev/gemfire/gemfire/gemfire-gfsh/build/distributed
Test/test-worker-000001/dunit/vm2/deployments
    os.version = 10.16
    path.separator = :
```

```
    sun.arch.data.model = 64
    sun.boot.class.path = /Library/Java/JavaVirtualMachines/liberica-jdk-8.jdk/Contents/Home/jre/lib/resources.jar:/Library/Java/JavaVirtualMachines/liberica-jdk-8.jdk/Contents/Home/jre/lib/rt.jar:/Library/Java/JavaVirtualMachines/liberica-jdk-8.jdk/Contents/Home/jre/lib/sunrsasign.jar:/Library/Java/JavaVirtualMachines/liberica-jdk-8.jdk/Contents/Home/jre/lib/jsse.jar:/Library/Java/JavaVirtualMachines/liberica-jdk-8.jdk/Contents/Home/jre/lib/jce.jar:/Library/Java/JavaVirtualMachines/liberica-jdk-8.jdk/Contents/Home/jre/lib/charsets.jar:/Library/Java/JavaVirtualMachines/liberica-jdk-8.jdk/Contents/Home/jre/lib/jfr.jar:/Library/Java/JavaVirtualMachines/liberica-jdk-8.jdk/Contents/Home/jre/classes
    sun.boot.library.path = /Library/Java/JavaVirtualMachines/liberica-jdk-8.jdk/Contents/Home/jre/lib
    sun.cpu.endian = little
    sun.cpu.isalist =
    sun.io.unicode.encoding = UnicodeBig
    sun.java.command = org.jboss.modules.Main -mp /Users/nnag/Development/gemfireDev/gemfire/gemfire/gemfire-assembly/build/install/vmware-gemfire-distributed-testing/moduleDescriptors/gemfire-gfsh:/Users/nnag/Development/gemfireDev/gemfire/gemfire/gemfire-gfsh/build/distributedTest/test-worker-000001/dunit/vm2/deployments gemfire
    sun.java.launcher = SUN_STANDARD
    sun.jnu.encoding = UTF-8
    sun.management.compiler = HotSpot 64-Bit Tiered Compilers
    sun.os.patch.level = unknown
    user.country = US
    user.language = en
    user.timezone = America/Los_Angeles
Log4J 2 Configuration:
    jar:file:/Users/nnag/Development/gemfireDev/gemfire/gemfire/gemfire-log4j/build/libs/gemfire-log4j-10.1.0-build.0.jar!/log4j2.xml
### GemFire Properties defined with system property ###
validate-serializable-objects=true
### GemFire Properties defined with api ###
http-service-port=0
locators=localhost[20004]
max-wait-time-reconnect=5000
name=server-2
redundancy-zone=RACK1
### GemFire Properties using default values ###
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
cluster-configuration-dir=
cluster-ssl-ciphers=any
cluster-ssl-enabled=false
cluster-ssl-keystore=
cluster-ssl-keystore-password=
cluster-ssl-keystore-type=
cluster-ssl-protocols=any
cluster-ssl-require-authentication=true
cluster-ssl-truststore=
cluster-ssl-truststore-password=
conflate-events=server
conserve-sockets=false
delta-propagation=true
deploy-working-dir=/Users/nnag/Development/gemfireDev/gemfire/gemfire/gemfire-gfsh/build/distributedTest/test-worker-000001/dunit/vm2
disable-auto-reconnect=false
disable-jmx=false
disable-tcp=false
distributed-system-id=-1
distributed-transactions=false
durable-client-id=
durable-client-timeout=300
```

```
enable-cluster-configuration=true
enable-management-rest-service=true
enable-network-partition-detection=true
enable-time-statistics=true
enforce-unique-host=false
gateway-ssl-ciphers=any
gateway-ssl-enabled=false
gateway-ssl-keystore=
gateway-ssl-keystore-password=
gateway-ssl-keystore-type=
gateway-ssl-protocols=any
gateway-ssl-require-authentication=true
gateway-ssl-truststore=
gateway-ssl-truststore-password=
groups=
http-service-bind-address=
http-service-ssl-ciphers=any
http-service-ssl-enabled=false
http-service-ssl-keystore=
http-service-ssl-keystore-password=
http-service-ssl-keystore-type=
http-service-ssl-protocols=any
http-service-ssl-require-authentication=false
http-service-ssl-truststore=
http-service-ssl-truststore-password=
jmx-manager=false
jmx-manager-access-file=
jmx-manager-bind-address=
jmx-manager-hostname-for-clients=
jmx-manager-http-port=0
jmx-manager-password-file=
jmx-manager-port=1099
jmx-manager-ssl-ciphers=any
jmx-manager-ssl-enabled=false
jmx-manager-ssl-keystore=
jmx-manager-ssl-keystore-password=
jmx-manager-ssl-keystore-type=
jmx-manager-ssl-protocols=any
jmx-manager-ssl-require-authentication=true
jmx-manager-ssl-truststore=
jmx-manager-ssl-truststore-password=
jmx-manager-start=false
jmx-manager-update-rate=2000
load-cluster-configuration-from-dir=false
locator-wait-time=0
lock-memory=false
log-disk-space-limit=0
log-file=
log-file-size-limit=0
log-level=config
max-num-reconnect-tries=3
mcast-address=239.192.81.1
mcast-flow-control=1048576, 0.25, 5000
mcast-port=0
mcast-recv-buffer-size=1048576
mcast-send-buffer-size=65535
mcast-ttl=32
member-timeout=5000
membership-port-range=41000-61000
memcached-bind-address=
memcached-port=0
memcached-protocol=ASCII
off-heap-memory-size=
remote-locators=
```

```
remove-unresponsive-client=false
roles=
security-auth-token-enabled-components=
security-client-accessor=
security-client-accessor-pp=
security-client-auth-init=
security-client-authenticator=
security-client-dhalgo=
security-log-file=
security-log-level=config
security-manager=
security-peer-auth-init=
security-peer-authenticator=
security-peer-verifymember-timeout=1000
security-post-processor=
security-udp-dhalgo=
serializable-object-filter=!*
server-bind-address=
server-ssl-ciphers=any
server-ssl-enabled=false
server-ssl-keystore=
server-ssl-keystore-password=
server-ssl-keystore-type=
server-ssl-protocols=any
server-ssl-require-authentication=true
server-ssl-truststore=
server-ssl-truststore-password=
socket-buffer-size=32768
socket-lease-time=1800000
ssl-ciphers=any
ssl-client-protocols=
ssl-cluster-alias=
ssl-default-alias=
ssl-enabled-components=
ssl-endpoint-identification-enabled=false
ssl-gateway-alias=
ssl-jmx-alias=
ssl-keystore=
ssl-keystore-password=
ssl-keystore-type=
ssl-locator-alias=
ssl-parameter-extension=
ssl-protocols=any
ssl-require-authentication=true
ssl-server-alias=
ssl-server-protocols=
ssl-truststore=
ssl-truststore-password=
ssl-truststore-type=
ssl-use-default-context=false
ssl-web-alias=
ssl-web-require-authentication=false
start-dev-rest-api=false
start-locator=
statistic-archive-file=
statistic-sample-rate=1000
statistic-sampling-enabled=true
tcp-port=0
thread-monitor-enabled=true
thread-monitor-interval-ms=60000
thread-monitor-time-limit-ms=120000
udp-fragment-size=60000
udp-recv-buffer-size=1048576
udp-send-buffer-size=65535
```

```
use-cluster-configuration=true
user-command-packages=

ClassLoader Modules:

  * Module Name: default_application:10.2.0-build.0

-- Resources:
--- /Users/nnag/Development/tcpWork/gemfire/gemfire-assembly/build/install/vmware-gemfire-distributed-testing/moduleDescript
ors/gemfire-gfsh/default_application/10.2.0-build.0/lib
-- Dependencies:
--- module java.base
--- local resources
```

---

describe offline-disk-store

Display information about an offline member's disk store.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
describe offline-disk-store --name=value --disk-dirs=value(,value)* [--pdx=value] [--region=value]
```

Parameters, describe offline-disk-store:

Name	Description
--name	*Required.* Name of the disk store to be described.
--disk-dirs	*Required.* Directory that contains the disk store files.
--pdx	If set (or set to true), display all the pdx types stored in the disk store.
--region	Name and path of the region in the disk store to be described.

Example Commands:

```
describe offline-disk-store --name=DiskStore1\
 --disk-dirs=/home/username/gemfire/mydiskStore1Dir

describe offline-disk-store --name=DiskStore1 --disk-dirs=/DiskDir1  --pdx=true
```

Sample Output:

```
gfsh>describe offline-disk-store --name=DiskStore1 --disk-dirs=/DiskDir1 --pdx=true

Regions in the disk store:
  /PdxTypes: -lru=none -concurrencyLevel=16 -initialCapacity=16 -loadFactor=0.75 -compressor=none -statisticsEnabled=false
  /Region1: -lru=none -concurrencyLevel=16 -initialCapacity=16 -loadFactor=0.75 -compressor=none -statisticsEnabled=false
PDX Types:
  com.app.data.PositionPdx: id=1
    long avg20DaysVol;
    String bondRating;
    double convRatio;
    String country;
    double delta;
    long industry;
    long issuer;
    double mktValue;
    double qty;
    String secId; // identity
    String secIdIndexed;
    String secLinks;
    double sharesOutstanding;
    String underlyer;
    long volatility;
    int pid;
    int portfolioId;
  com.app.data.StockPdx: id=2
    int ID; // identity
    String pkid;
    Object position1;
    Object position2;
    Object positions;
    Object collectionHolderMap;
    String type;
    String status;
    String[] names;
    String description;
    long createTime;
    Object[] position3;
    Object aDay;
    Date date;
PDX Enums:
  com.app.data.StockPdx$Day.Monday
```

---

describe query

Display additional details about an executing query.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
describe query --query-id=value [--(member|members)=value(,value)*]
```

Parameters, describe query:

Name	Description
--query-id	*Required.* ID of the query. To find a client ID, use the list queries command to display a list of executing queries and their IDs.
--member or --members	Name or ID of the members for which the query will be displayed. If this is not specified then the queries from all members will be displayed.

Example Commands:

```
describe query --queryId=faf13b3b-0da1-4b75-885f-2dc1a010f080
describe query --queryId=faf13b3b-0da1-4b75-885f-2dc1a010f080 --members=server1,server3
```

Sample Output:

```
gfsh>describe query --queryId=faf13b3b-0da1-4b75-885f-2dc1a010f080

Query ID          : fee0909a-75b6-4fd5-8ccb-4f1b666c5d42
Query             : <TRACE> SELECT * FROM /PORTFOLIO p WHERE p.ID >= 0
Elapsed Time      : 00:00:00.153
Execution Count   : 2
Avg. Execution Time : 00:00:00.009
Indexes Used      : PORTFOLIO_IDX*
Members           : server-1, server-2
[Time format is hh:mm:ss.sss. Indexes marked with (*) were not found in use on all members.]
```

Elapsed Time shown is the maximum elapsed time in seconds from all members where the query is being executed, or for the members as specified by the --member or --members arguments.

Indexes Used is the union of the names of the indexes that have been used for this queries on all of the relevant members.

---

describe query-service

Print the information that describes the current configuration of the query service.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
describe query-service
```

Example Command:

```
describe query-service
```

Sample Output:

```
gfsh>describe query-service

Method Authorizer Class : org.apache.geode.cache.query.security.JavaBeanAccessorMethodAuthorizer
```

---

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 [--verbose]
```

Parameters, describe region:

Name	Description
--name	*Required.* Name or path of the region to be described.
--verbose	Optional. Displays additional properties of the region.

Output:

Output Field	Description
Name	Name of the region.
Data Policy	Specifies how the local cache handles data for a region. This setting controls behavior such as local data storage and region initialization. Possible Values: • Empty: No data storage in the local cache. • Normal: Data used locally is stored in the local cache. • Partition: Data is partitioned across local and remote caches using the automatic data distribution behavior of partitioned regions. • Replicate: The region is initialized with the data from other caches. • Persistent-partition: Behaves the same as partition and also persists data to disk. • Persistent-replicate: Behaves the same as replicate and also persists data to disk. • Preloaded: Initializes like a replicated region. After initialization, behaves like a normal region.
Hosting Members	Members hosting the region and its entries.
disk-store-name	Name of the disk store the region is assigned to.
size	Number of entries in the region.
scope	Describes the policy for region distribution. • distributed-no-ack • distributed-ack • global
Accessor Members	Members hosting the region but not its entries. These members are set with memory value set to zero and cannot store actual data.
concurrency-checks-enabled	Determines whether members perform checks to provide consistent handling for concurrent or out-of-order updates to distributed regions. For more information, see Consistency Checking by Region Type.
pool-name	Server pool used by the region.
enable-subscription-conflation	Whether or not the server conflates region's messages sent to the client. • true: Server conflates region's messages sent to the client. • false: No conflation.

Output Field	Description
entry-time-to-live.action	Action that should take place when an entry expires. Possible values: • local-destroy • destroy • local-invalidate • invalidate
gateway-sender-id	All gateway senders associated with the region.
region-time-to-live.action	Action that should take place when a region expires. Possible values: • local-destroy • destroy • local-invalidate • invalidate
region-idle-time.action	Action that should take place when a region remains idle until the set timeout. Possible values: • local-destroy • destroy • local-invalidate • invalidate
region-idle-time.timeout	Time, in seconds, that the region can remain in the cache without being accessed or updated. Deactivated indicates no expiration.
initial-capacity	The initial capacity (entries) of the region's underlying java.util.ConcurrentHashMap used for storing region entries
statistics-enabled	Whether or not the statistics are enabled. • true: Region statistics are enabled. • false: Region statistics are disabled and expiration cannot be enabled.
cache-loader	Fully-qualified class name of the region's cache loaders.
cache-listeners	Fully-qualified class name of the region's cache listeners.
concurrency-level	Maximum number of application threads that can concurrently access the region's entry.
entry-time-to-live.timeout	Time, in seconds, for which a region's entries can remain in the cache. Deactivated indicates no expiration.
index-maintenance-synchronous	Whether or not indexes are maintained synchronously with region modifications. • true: Indexes are maintained synchronously with region modifications. • false: Indexes are maintained asynchronously in a background thread.
entry-idle-time-custom-expiry	Fully-qualified class name for the action to perform when a region's entries idle time expires.
load-factor	The load factor, expressed as a percentage in decimal format, pertains to the region's java.util.ConcurrentHashMap, which stores region entries. It determines when the ConcurrentHashMap's capacity should be increased.

Output Field	Description
cloning-enabled	Whether or not cloning is enabled. Determines how value changes are made in the cache. • true: Updates are applied to a clone of the value and then the clone is saved to the cache. • false: Value is modified in place in the cache.
off-heap	Whether or not off-heap memory is used to store region values and queue entries. • true: Off-heap memory used to store the region values and queue entries. • false: Off-heap memory not used.
enable-async-conflation	Whether or not asynchronous messages for the region are conflated by the sending server. • true: Asynchronous messages for the region are conflated by the sending server. • false: No conflation.
is-lock-grantor	Whether or not the regions is a lock grantor. • true: Region is a lock grantor. • false: Region cannot grant lock.
entry-idle-time.timeout	Time, in seconds, that the region's entries can remain in the cache without being accessed or updated. Deactivated indicates no expiration.
region-time-to-live.timeout	Time, in seconds, before the region expires. Deactivated indicates no expiration.
ignore-jta	Whether or not the regions participates in JTA transactions. • true: Region participates in JTA transactions. • false: Region ignores JTA and operates outside of transactions.
entry-idle-time.action	Action to perform when the region's entries remain idle for the timeout duration. Possible values: • local-destroy • destroy • local-invalidate • invalidate
async-event-queue-id	Asynchronous event queue identifiers associated with the region.
cache-writer	Fully-qualified class name of the region's cache writer.
value-constraint	Allowed data type for region entry values.
key-constraint	Allowed data type for region entry keys.
disk-synchronous	Whether or not disk writes are synchronous. • true: Disk writes are synchronous. • false: Disk writes are asynchronous.
compressor	Fully-qualified class name of the Compressor to use when compressing region entry values.
object-sizer	Fully-qualified class name used to calculate the size of the region.
eviction-algorithm	Algorithm used to determine when to start evicting entries.

Output Field	Description
eviction-action	Action triggered by the eviction-algorithm decision.
eviction-maximum-value	• If the eviction-algorithm is based on the number of entries, this value is the maximum number of entries. • If the eviction-algorithm is based on memory usage, this value is the maximum allowed memory in MB. • If the eviction-algorithm is based on the percentage of heap in use, this value is always zero.
redundant-copies	Number of secondary copies maintained in the region's data stores.
colocated with	Name of the region this is co-located with.
partition-resolver	Partition resolver used to customize the default hashing policy for distributing entries into buckets.
total-num-buckets	Total number of buckets in the region.
number of buckets with redundancy satisfied	Number of buckets in the region which have the same redundancy as this region was configured with.
partition-listeners	Partition listeners associated with the region.
number of buckets with redundancy not satisfied	Number of buckets in the region which have lower redundancy than this region was configured with.
startup-recovery-delay	Time, in milliseconds, to wait after the member joins and before recovering redundancy. Deactivated indicates no automatic recovery of redundancy after a new member comes online.
redundancy	• Satisfied: All buckets have the configured redundancy. • Not Satisfied: One or more buckets do not have redundancy equal to the configured value.
entry-time-to-live-custom-expiry	Fully-qualified class name for the action to perform when a region's entry time to live expires.
recovery-delay	Time, in milliseconds, that existing members will wait before satisfying redundancy after another member crashes. Deactivated indicates that redundancy will not be recovered after a failure.
local-max-memory	Maximum amount of memory, in megabytes, to be used by the region in this process. If not set, a default of 90% of available heap is used.
Fixed Partition	Fixed custom partitioning is deprecated and will be removed in upcoming releases. Fixed partition attributes associated with the region: • Name • Primary/Secondary • Number of buckets
number of entries with values only on disk	Number of values in the region that have overflowed to disk and are no longer in memory.
total size of values only on disk	Total size of the values that have overflowed to disk.
number of entries with values in memory	Number of entries in the JVM memory.
total size of the region in memory	Size of the region in the JVM memory.

Example Commands:

```
describe region --name=region1
describe region --name=PR1 --verbose
```

Sample Output 1:

```
gfsh>describe region --name=Region1

..........................................................
Name : Region1
Data Policy : persistent replicate
Hosting Members : server-5
server-4
server-3
server-2

Non-Default Attributes Shared By Hosting Members

Type   | Name            | Value
------ | --------------- | --------------------
Region | data-policy     | PERSISTENT_REPLICATE
       | disk-store-name | DiskStore1
       | size            | 0
       | scope           | distributed-ack

..........................................................
Name             : Region1
Data Policy      : empty
Accessor Members : server-1

Non-Default Attributes Shared By Accessor Members

 Type  |    Name     | Value
------ | ----------- | ---------------
Region | data-policy | EMPTY
       | size        | 0
       | scope       | distributed-ack
```

Sample Output 2:

```
gfsh>describe region --name=PR1 –verbose

Name            : PR1
Size            : 0
Data Policy     : partition
Status          : Ready
Hosting Members : server-1
                  server-2

All Attributes Shared By Hosting Members

  Type    |                    Name                    | Value
--------- | ---------------------------------------------- | -----------------
Region    | concurrency-checks-enabled                 | true
          | pool-name                                  |
          | enable-subscription-conflation             | false
          | entry-time-to-live.action                  | INVALIDATE
          | gateway-sender-id                          |
          | region-time-to-live.action                 | INVALIDATE
          | region-idle-time.action                    | INVALIDATE
          | disk-store-name                            |
          | region-idle-time.timeout                   | Deactivated
          | initial-capacity                           | 16
          | statistics-enabled                         | true
          | cache-loader                               |
          | cache-listeners                            |
          | concurrency-level                          | 4
          | entry-time-to-live.timeout                 | Deactivated
          | data-policy                                | PARTITION
          | index-maintenance-synchronous              | true
          | entry-idle-time-custom-expiry              | TestCustomIdleExpiry
          | load-factor                                | 0.75
          | cloning-enabled                            | false
          | off-heap                                   | false
          | enable-async-conflation                    | true
          | is-lock-grantor                            | false
          | entry-idle-time.timeout                    | Deactivated
          | region-time-to-live.timeout                | Deactivated
          | ignore-jta                                 | false
          | entry-idle-time.action                     | INVALIDATE
          | async-event-queue-id                       |
          | cache-writer                               |
          | value-constraint                           |
          | key-constraint                             |
          | disk-synchronous                           | true
          | compressor                                 |
Eviction  | object-sizer                               |
          | eviction-algorithm                         | lifo-entry-count
          | eviction-action                            | local-destroy
          | eviction-maximum-value                     | 100
Partition | redundant-copies                           | 1
          | colocated with                             |
          | partition-resolver                         |
          | total-num-buckets                          | 113
          | number of buckets with redundancy satisfied | 0
          | partition-listeners                        | TestPartitionListener
          | number of buckets with redundancy not satisfied | 0
          | startup-recovery-delay                     | 0 ms
          | redundancy                                 | Satisfied

All Attributes Specific To The Hosting Members

 Member  |   Type    |              Name              | Value
```

```
-------- | --------- | ------------------------------- | --------------------
server-1 | Region    | entry-time-to-live-custom-expiry |
         | Partition | recovery-delay                   | 2 ms
         |           | local-max-memory                 | 100 MB
         |           | Fixed Partition                  | Par1
         |           |                                  | Primary
         |           |                                  | 1
server-2 | Region    | entry-time-to-live-custom-expiry | TestCustomTTLExpiry
         | Partition | recovery-delay                   | 4 ms
         |           | local-max-memory                 | 150 MB
         |           | Fixed Partition                  | Par2
         |           |                                  | Secondary
         |           |                                  | 4

Region Entries Statistics Specific To The Hosting Members

 Member  |     Type      |                Name                | Value
-------- | ------------- | ---------------------------------- | -------------
server-1 | Disk Overflow | number of entries only on disk      | Not Available
         |               | number of bytes overflowed to disk  | Not Available
         |               | number of entries in the cache      | 0
         |               | number of bytes stored in the cache | 0 bytes
server-2 | Disk Overflow | number of entries only on disk      | Not Available
         |               | number of bytes overflowed to disk  | Not Available
         |               | number of entries in the cache      | 0
         |               | number of bytes stored in the cache | 0 bytes
```

---

describe wan-copy region

Use this command to provide detailed information about a running wan-copy region command. For more information, see wan-copy region.

Syntax

```
describe wan-copy region --wan-copy-id=wan-copy-id [--verbose]
```

Parameters

--wan-copy-id: Specifies the wan-copy-id of the wan-copy region command to be described (required). --verbose: A flag that specifies whether to provide detailed information (optional).

Output

If the specified wan-copy-id exists, the command will print a header with the total number of regions being copied and the duration of the wan-copy process. If the –verbose flag is not set, it will display a table containing each region/sender ID pair being copied and their status. If the –verbose flag is set, the command will provide a detailed header and table for the region/sender pairs showing the current status of each member.

Examples 1

```
gfsh>describe wan-copy region --wan-copy-id=cc09e6a1-99a7-4515-ba7c-3ec2caa15446 --verbose

3 of 6 regions copied.
Elapsed Time: 2 hrs 45 min 33 sec

Region Name: region1-east
Sender Id: ny-east
Status: FINISHED

Member      | Status | Message
----------- | ------ | -------------------
server-ln-2 | OK     | Entries copied: 252
server-ln-1 | OK     | Entries copied: 248

Region Name: region1-west
Sender Id: ny-west
Status: FINISHED

Member      | Status | Message
----------- | ------ | -------------------
server-ln-2 | OK     | Entries copied: 85
server-ln-1 | OK     | Entries copied: 85


Region Name: region1-north
Sender Id: ny-north
Status: FINISHED

Member      | Status | Message
----------- | ------ | -------------------
server-ln-2 | OK     | Entries copied: 120
server-ln-1 | OK     | Entries copied: 120


Region Name: region1-south
Sender Id: ny-south
Status: FINISHED

Member      | Status | Message
----------- | ------ | -------------------
server-ln-2 | OK     | Entries copied: 252
server-ln-1 | OK     | Entries copied: 248


Region Name: region1-central
Sender Id: ny-central
Status: PROGRESSING

Member      | Status | Message
----------- | ------ | -------------------
server-ln-2 | OK     | Entries copied: 252
server-ln-1 | OK     | In Progress
```

Example 2

```
gfsh>describe wan-copy region --wan-copy-id=cc09e6a1-99a7-4515-ba7c-3ec2caa15446

3 of 6 regions copied.
Elapsed Time: 2 hrs 45 min 33 sec

Region Name     | Sender-Id    Status
----------------|--------------|----------
region1-east    | ny-east      | FINISHED
region1-west    | ny-west      | FINISHED
region1-north   | ny-north     | FAILED
region1-south   | ny-south     | FINISHED
region1-central | ny-central   | IN PROGRESS
region1-PR      | ny-pr        | NOT STARTED
```

destroy

This topic describes the destroy command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to delete or unregister functions, or to remove indexes, disk stores, and regions.

- destroy async-event-queue

  Destroy an asynchronous event queue.

- destroy disk-store

  Delete a disk store and all files on disk used by the disk store.

- destroy function

  Destroy or unregister a function.

- destroy gateway-receiver

  Destroy a gateway receiver.

- destroy gateway-sender

  Destroy a gateway sender.

- destroy index

  Destroy or remove the specified index.

- destroy initializer

  Close and destroy configuration for the initializer for the specified groups. Cluster configuration must be enabled to use this command.

- destroy jndi-binding

  Destroy the specified JNDI binding.

- destroy region

  Destroy or remove a region.

destroy async-event-queue

Destroy an asynchronous event queue.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
destroy async-event-queue --id=value [--groups=value(,value)*] [--if-exists=value]
```

Parameters, destroy async-event-queue:

Name	Description
--id	*Required.* ID of the async event queue to be deleted.
--groups	Groups of members on which the async event queue will be destroyed. If no group is specified, the queue is destroyed on all members.
--if-exists	If the specified async event queue does not exist, gfsh responds with a message to that effect. If this parameter is true, the response is prefixed with the label "Skipping: ". Useful for scripted tests. Default (if the parameter is not specified): false. Default (if the parameter is specified without value): true.

Example Commands:

```
destroy async-event-queue --id=myAsyncEventQueue
```

destroy disk-store

Delete a disk store and all files on disk used by the disk store. Data for closed regions that previously used this disk store are lost.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
destroy disk-store --name=value [--groups=value(,value)*] [--if-exists=value]
```

Parameters, destroy disk-store:

Name	Description
--name	*Required.* Name of the disk store to be deleted.
--groups	Groups of members on which the disk store will be destroyed. If no group is specified, the disk store is destroyed on all members.
--if-exists	If the specified disk store does not exist, gfsh responds with a message to that effect. If this parameter is true, the response is prefixed with the label "Skipping: ". Useful for scripted tests. Default (if the parameter is not specified): false. Default (if the parameter is specified without value): true.

Example Commands:

```
destroy disk-store --name=store1
```

Sample Output:

```
gfsh>destroy disk-store --name=store1
Member  | Result
------- | -------
server1 | Success
```

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 [--groups=value(,value)*] [--member=value]
```

Parameters, destroy function:

Name	Description
--id	*Required.* Unique function identifier. Use the list functions command to obtain the ID.
--groups	One or more groups of members from which this function will be unregistered.
--member	Name or 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 gateway-receiver

Destroy the gateway receiver on one or more members. Since there can be only one gateway receiver per member, specifying the member specifies which gateway receiver to destroy.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
destroy gateway-receiver [--group=value(,value)*]
  [--member=value(,value)*] [--if-exists(=value)?]
```

Parameters, destroy gateway-receiver:

Name	Description
--group	One or more groups of members on which the gateway receiver will be destroyed.
--members	Name or ID of each member on which the gateway receiver will be destroyed.
--if-exists	If this option is true and the gateway receiver does not exist, gfsh responds with a message prefixed with the label "Skipping: ". If this option is false and the gateway receiver does not exist, an error message is output. Default (if this option is not specified): false. Default (if this option is specified without a value): true.

Example Commands:

```
destroy gateway-receiver --member=server1,server2
```

destroy gateway-sender

Destroy a gateway sender that is no longer used by a region.

The default is for the gateway sender to be destroyed on all members.

No region may be attached to the gateway sender to be destroyed. If a region is still attached, the system issues an error message similar to:

```
ERROR: The GatewaySender ParallelGatewaySender{id=ln,remoteDsId=2,isRunning =false}
could not be destroyed as it is still used by regions.
```

Remove the gateway sender from the region with a command similar to:

```
gfsh>alter region --name=regionA --gateway-sender-id=""
```

> Do not reuse the removed parallel gateway-sender in the new region unless that region is colocated with the previously attached region, as that will lead to data replication issues.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
destroy gateway-sender --id=value [--groups=value(,value)*]
  [--members=value(,value)*] [--if-exists=value]
```

Parameters, destroy gateway-sender:

Name	Description
--id	*Required.* Unique gateway sender identifier. Use the list gateways command to obtain the ID.
--groups	One or more groups of members from which this gateway sender will be destroyed.
--members	Name or ID of the members from which this gateway sender will be destroyed.
--if-exists	If the specified gateway sender does not exist, gfsh responds with a message to that effect. If this parameter is true, the response is prefixed with the label "Skipping: ". Useful for scripted tests. Default (if the parameter is not specified): false. Default (if the parameter is specified without value): true.

Example Commands:

```
destroy gateway-sender --id=SiteASender
```

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] [--members=value(,value)*]
[--groups=value(,value)*] [--if-exists=value]
```

> 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.

Parameters, destroy index:

Name	Description
--name	Name of the index to be removed.
--members	Id of the members on which index is to be removed.
--region	Name of the region from which an index or all indexes are to be destroyed.
--groups	The index will be removed on all the members in the groups.
--if-exists	If the specified index does not exist, gfsh responds with a message to that effect. If this parameter is true, the response is prefixed with the label "Skipping: ". Useful for scripted tests. Default (if the parameter is not specified): false. Default (if the parameter is specified without value): true.

Example Commands:

```
destroy index --members=server2
destroy index --name=MyKeyIndex
```

destroy initializer

Close and destroy configuration for the initializer for the specified groups. If the initializer class has implemented the CacheCallback interface, the close method will be called. Cluster configuration must be enabled to use this command.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
destroy initializer [--groups=value(,value)*]
```

Parameters, destroy initializer:

Name	Description
--groups	Groups for which the initializer will be closed and destroyed. If no groups are specified, the cluster group is assumed.

Example Commands:

```
destroy initializer --groups=group1,group2
destroy initializer
```

destroy jndi-binding

Destroy a specified JNDI binding that holds the configuration for an XA data source.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
destroy jndi-binding --name=value [--if-exists=value]
```

Parameters, destroy jndi-binding:

Name	Description
--name	*Required.* Name of the JNDI binding to be destroyed.

Name	Description
--if-exists	Skip the destroy operation when the specified JNDI binding does not exist. Without this option, an error results from the specification of a JNDI binding that does not exist. Default (if the parameter is not specified): false. Default (if the parameter is specified without value): true.

Example Command:

```
destroy jndi-binding --name=jndi1
```

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 [--if-exists=value]
```

Parameters, destroy region:

Name	Description
--name	*Required.* Name and path of the region to be removed.
--if-exists	If the specified region does not exist, gfsh responds with a message to that effect. If this parameter is true, the response is prefixed with the label "Skipping: ". Useful for scripted tests. Default (if the parameter is not specified): false. Default (if the parameter is specified without value): true.

Example Commands:

```
destroy region --name=region4
destroy region --name=/region1/subregion1
```

Sample Output:

```
gfsh>destroy region --name=region1
"region1"  destroyed successfully.
```

disconnect

This topic describes the disconnect command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to close any active connections.

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: Locator1[1099]
Disconnected from : Locator1[1099]
```

Error Messages:

```
Error occurred while disconnecting: {0}
```

```
Not connected!
```

echo

This topic describes the echo command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to echo the specified text. This text may include system and user variables.

The command can also echo gfsh environment properties (using 'set variable' command) if the variable name is pre-pended with '$'.

For a list of gfsh environment variables, see Useful gfsh Shell Variables.

Availability: Online or offline.

Syntax:

```
echo [--string=value]
```

Name	Description
--string	String to be echoed. For example, SYS_USER variable is set to ${SYS_USER}.

Table 1. Echo Parameters

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:

```
gfsh>echo --string=${SYS_JAVA_VERSION}
Post substitution: echo --string=1.8.0_60
1.8.0_60
```

execute function

This topic describes the execute command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to execute functions on members or regions.

execute function

This command executes the function with the specified ID. By default, the specified function executes 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 [--groups=value(,value)*]
[--members=value(,value)*] [--region=value]
[--arguments=value] [--result-collector=value] [--filter=value]
```

--id	*Required.* ID of the function to execute.
--groups	One or more groups of members on which this function should be executed.
--members	Name/ID of the members on which the function will be executed.
--region	Region on which the data dependent function will be executed.
--arguments	Arguments to the function in comma-separated string format.
--result-collector	Fully qualified class name of the ResultCollector to instantiate for gathering results.
--filter	Key list which causes the function to only be executed on members which have entries with these keys.

Table 1. Execute Function Parameters

Example Commands:

```
execute function --id=InterestCalculations --region=/InterestRegion
execute function --id=InterestCalculations --members=server1
execute function --id=InterestCalculations --groups=Group1
```

> The execute function passes the value of --arguments as a String Array to any executed function. This differs from the behavior of FunctionExecutionService API. FunctionExecutionService API passes the value of the argument as an Object.

exit

This topic describes the exit command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to exit the gfsh shell and return to the OS shell. You can also use the quit command to exit the gfsh shell and return to the OS shell.

Availability: Online or offline.

Syntax:

```
exit
```

Example Commands:

```
exit
```

export

This topic describes the export command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to export configurations, data, logs, and stack-traces.

- export cluster-configuration

  Export a cluster configuration ZIP file that contains the cache.xml files, gemfire.properties files, and application JAR files needed to configure and operate a cluster.

- export config

  Export configuration properties for a member or members.

- export data

  Export user data from a region to a file.

- export logs

  Export logs to a given directory.

- export offline-disk-store

  Export region data from an offline disk store into gemfire snapshot files.

- export stack-traces

  Export the stack trace for a member or members.

export cluster-configuration

Exports a single XML file or a ZIP file with cluster configuration that contains the cache.xml files, gemfire.properties files, and application JAR files needed to configure and operate a cluster.

When neither a file name nor a ZIP file name is specified, the cluster configuration is written to standard output.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

See Overview of the Cluster Configuration Service.

Syntax:

```
export cluster-configuration [--group(=value)?] [--xml-file=value]
 [--zip-file-name=value]
```

Export Cluster-Configuration Parameters:

Name	Description	Default Value
--group	Export the configuration for the specified server group. When no group is specified, use the default cluster group.	
--xml-file	Filename to contain the exported cluster configuration. May also include an absolute or relative path. Only specify one of --xml-file or --zip-file-name.	
--zip-file-name	Filename of the ZIP file to contain the exported cluster configuration. May also include an absolute or relative path. Only specify one of --xml-file or --zip-file-name.	

Example Commands:

```
gfsh>export cluster-configuration --zip-file-name=/group/shared-configs/devClusterConfig.zip
gfsh>export cluster-configuration --zip-file-name=my-configs/myConfig.zip
gfsh>export cluster-configuration --zip-file-name=myClusterConfig.zip
gfsh>export cluster-configuration --xml-file=Cluster3Config.xml
```

Sample Output:

```
gfsh>export cluster-configuration --zip-file-name=mySharedConfig.zip
Downloading cluster configuration : /home/username/gemfire/mySharedConfig.zip
```

export config

Export configuration properties for a member or members.

If you do not specify any parameters, all member configurations will be exported.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
export config [--members=value(,value)*] [--groups=value(,value)*]
[--dir=value]
```

Export Config Parameters:

Name	Description
--members	Name/Id of the members whose configuration will be exported.
--groups	Groups of members whose configuration will be exported.
--dir	Directory to which the exported configuration files will be written.

Example Commands:

```
export config
export config --members=member1
```

Sample Output:

```
gfsh>export config --members=member1
Downloading Cache XML file: c:\VMwareGemFire\Latest\.\member1-cache.xml
Downloading properties file: c:\VMwareGemFire\Latest\.\member1-gf.properties
```

export data

Export user data from a region to a file or files.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

To export data from any region, replicated or partitioned, sequentially to a single file, use the --file option:

```
export data --region=value --member=value --file=value
```

In this scenario, data from replicated and partitioned regions is exported sequentially to a single file on the specified member.

In the case of partitioned regions, you can speed up the export process by using the --parallel option:

```
export data  --parallel --region=value --member=value --dir=value
```

In this scenario, partitioned region data is exported simultaneously on all hosting nodes to directories local to those respective nodes.

- Parallel export applies only to partitioned regions.
- The --file and --dir options are mutually exclusive; --file can be used only for serial export, --dir can be used for both serial and parallel exports.

Export Data Parameters:

Name	Description
--region	*Required.* Region from which data is to be exported.
--member	*Required.* Name/ID of a member that hosts the region. In a serial export, all data is exported to the specified file on the host where the member is running. In a parallel export, data from a partitioned region partially hosted on this member is exported for each partition to files on the nodes that host those partitions.
--file	File to which the exported data is to be written. The file must have an extension of .gfd. Cannot be specified at the same time as --dir, cannot be used with --parallel.
--dir	Directory to which the exported data is to be written. Required if --parallel is true. Cannot be specified at the same time as --file.
--parallel	Export local data on each node to a directory on that machine. Available for partitioned regions only.

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:\VMwareGemFire\
Latest\server1\region2_20121001.gfd on host : 192.0.2.0
```

export logs

Export 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.

The --dir parameter specifies a local directory to which log files will be written. This is used only when you are exporting logs using an http connection. If executed over http, the zip archive will be saved in the specified directory on the user's client machine. If not specified, logs are written to the location specified by the user.dir system property. When the command is executed over JMX, logs will be saved as exportedLogs_xxx.zip in the connected locator's working directory.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
export logs [--dir=value] [--groups=value(,value)*] [--members=value(,value)*]
[--log-level=value] [--only-log-level=value] [--merge-log=value]
[--start-time=value] [--end-time=value] [logs-only(=value)?]
[--stats-only(=value)?] [--file-size-limit(=value)?]
```

Export Logs Parameters:

Name	Description	Default Value
--dir	Local directory to which log files will be written when logs are exported using an http connection. Ignored when the command is executed over JMX.	
--groups	Groups of members whose log files will be exported.	
--members	Name/ID of the members whose log files will be exported.	
--log-level	Minimum level of log entries to export. Valid values are: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, and ALL.	INFO
--only-log-level	Whether to only include only entries that exactly match the --log-level specified.	false
--merge-log	Whether to merge logs after exporting to the target directory (deprecated).	false
--start-time	Log entries that occurred after this time will be exported. Format: yyyy/MM/dd/HH/mm/ss/SSS/z OR yyyy/MM/dd. When using the yyyy/MM/dd/HH/mm/ss/SSS/z format, the time zone, denoted z, should be specified as either a 3-letter time zone such as PST or as an offset to GMT/UTC such as GMT+08:00.	no limit
--end-time	Log entries that occurred before this time will be exported. Format: yyyy/MM/dd/HH/mm/ss/SSS/z OR yyyy/MM/dd. When using the yyyy/MM/dd/HH/mm/ss/SSS/z format, the time zone, denoted z, should be specified as either a 3-letter time zone such as PST or as an offset to GMT/UTC such as GMT+08:00. An end time specified by only a date implements a time of 00:00. This exports logs written up until 23:59:59.999 on the date prior to the one specified.	no limit
--logs-only	Whether to export only logs (not statistics)	If parameter not specified: false. If parameter specified without a value: true
--stats-only	Whether to export only statistics (not logs)	If parameter not specified: false. If parameter specified without a value: true
--file-size-limit	Limits total unzipped size of the exported files. Specify 0 (zero) for no limit. Value is in megabytes by default or [k,m,g,t] may be specified.	If parameter not specified: 100m. If parameter specified without a value: 0

Example commands, showing output:

```
gfsh>export logs --dir=data/logs
Logs exported to the connected member's file system: /my-locator/data/logs/exportedLogs_1489513007261.zip
```

```
gfsh>export logs --start-time=2020/12/14/12/00/00/000/GMT-08:00 --end-time=2020/12/27 --dir=data/logs
Start time parsed as 2020-12-14T12:00 PST
End time parsed as 2020-12-27T00:00 PST
Logs exported to the connected member's file system: /my-locator/data/logs/exportedLogs_1608165308676.zip
```

```
gfsh>export logs --dir=data/logs --file-size-limit=1k
Estimated exported logs expanded file size = 95599, file-size-limit = 1024.
To deactivate exported logs file size check use option "--file-size-limit=0".

gfsh>export logs --dir=data/logs --file-size-limit=99k
Logs exported to the connected member's file system: /my-locator/data/logs/exportedLogs_1489513007261.zip
```

export offline-disk-store

Export region data from an offline disk store into GemFire snapshot files.

Availability: Online or offline.

Syntax:

```
export offline-disk-store --name=value --disk-dirs=value(,value)* --dir=value
```

Export Offline-Disk-Store Parameters:

Name	Description
--name	*Required.* Name of the disk store to be exported.
--disk-dirs	Directories which contain the disk store files.
--dir	Directory to export the snapshot files to.

Example Commands:

```
 export offline-disk-store --name= DiskStore1 \
--disk-dirs=/home/username/gemfire/mydiskStore1Dir --dir=/home/username/gemfire/export
```

export stack-traces

Export the stack trace for one or more cluster members (locators and servers).

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
export stack-traces [--members=value(,value)*] [--groups=value(,value)*]
[--file=value] [--abort-if-file-exists(=value)?]
```

Export Stack-Traces Parameters:

Name	Description
--members	Name or ID of the members whose stack traces will be exported.
--groups	Groups of members whose stack traces will be exported.
--file	File name to which the stack traces will be written. When not specified, the file name will be the string “stacktrace_” followed by the current time in milliseconds.
--abort-if-file-exists	When true, abort the command if the file already exists in the locator’s directory. If this option is not specified, the default value is false.

Example Commands:

```
export stack-traces --file=stack.txt --abort-if-file-exists=true
```

Sample Output:

```
gfsh>export stack-traces
stack-traces exported to file: /data/stacktrace_1612316330340
On host : cluster-locator-0.cluster-locator.system.svc.cluster.local
```

gc

This topic describes the gc command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to force garbage collection (GC) on a member or members.

The default is for garbage collection to occur on all caching members.

The gfsh gc command should be used only in non-production environments. This is a forced GC event and not always handled well by the garbage collector outside of normal GC processing. This can cause slow-responding members to be perceived as unresponsive and disconnected from the cluster.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
gc [--groups=value(,value)*] [--member=value]
```

Name	Description
--groups	One or more groups of members on which garbage collection will be forced.
--member	Name/ID of the member on which garbage collection will be forced.

Table 1. GC Parameters

Example Commands:

```
gc --member=server1
gc --groups=Group1
gc
```

Sample Output:

```
gfsh>gc
Successfully executed GC
```

get

This topic describes the get command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to 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]
```

Name	Description	Default Value
--key	*Required.* String or JSON text from which to create the key. For example: “James”, “100L” and “('id': 'l34s')”.	

Name	Description	Default Value
--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.
--load-on-cache-miss	Explicitly activates or deactivate the use of any registered CacheLoaders on the specified Region when retrieving a value for the specified Key on Cache misses.	true (enabled)

Table 1. Get Parameters

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

This topic describes the help command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to display syntax and usage information a gfsh command. If a command is not specified, this command lists all available gfsh commands.

Availability: Online or offline.

Syntax:

```
help [command]
```

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

This topic describes the hint command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to display information about topics and a list of commands associated with a topic. If a topic is not specified, this command lists all available topics.

Availability: Online or offline.

Syntax:

```
hint [topic]
```

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

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

This topic describes the history command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to show or save the command history.

You can save this history to a file which you can later use as a script.

A history of commands that have been executed successfully is also logged in the .geode/.gfsh.history file within the home directory of the user running gfsh.

Availability: Online or offline.

Syntax:

```
history [--file=<history text file>]
```

Name	Description	Default Value
--file	File to which the history is to be saved.	
--clear	When set to true, clears the history of gfsh commands.	false

Table 1. History Parameters

Example Commands:

```
history
history --file=./mycommands.gfsh;
```

Sample Output:

```
gfsh>history --file=./mycommands.gfsh
Wrote successfully to file ./mycommands.gfsh
```

import

This topic describes the import command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to import an exported cluster configuration to create a new cluster or import data into a region.

- import cluster-configuration

  Import a cluster configuration.

- import data

  Import user data from a file to a region.

import cluster-configuration

Imports a previously exported cluster configuration from a ZIP file or an XML file. This command is useful when spinning up a new cluster.

- In a given cluster, only one locator needs to perform the import. That locator shares the imported configuration with all other connected locators in the cluster.
- Locators share the imported configuration with data members (servers) when the servers start, or if the servers have been recently started, have no regions defined in them, and have been given no other configuration changes since they started.

To import a cluster configuration, start one or more locators and then run the gfsh import cluster-configuration command.

Availability: Online. You must be connected in gfsh to a locator to use this command.

Syntax:

```
import cluster-configuration [--action=value] [--group(=value)?]
 [--xml-file=value] [--zip-file-name=value]
```

Import Cluster-Configuration Parameters:

Name	Description
--action	When the value is APPLY (the default), the configuration is applied to the running servers that have no configuration. If any servers already have configuration, the command fails. When the value is STAGE, the configuration is overwritten and will be used during future server creation; the configuration of currently running servers is not changed.
--group	Do the import for the specified server group. When no group is specified, the cluster is implied.
--xml-file	Filename from which to import the cluster configuration. May also include an absolute or relative path. Only specify one of --xml-file or --zip-file-name.
--zip-file-name	Name of the ZIP file containing the cluster configuration artifacts to be imported. Only specify one of --xml-file or --zip-file-name.

Example Commands:

```
gfsh>import cluster-configuration --zip-file-name=/home/username/myConfig.zip
gfsh>import cluster-configuration --xml-file=configs/Cluster3Config.xml
```

Sample Output:

```
gfsh>import cluster-configuration --zip-file-name=/home/username/myConfig.zip
Cluster configuration successfully imported
```

import data

Import user data from a file or files to a region.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

If the data was exported sequentially to a single file, use the serial form of the import command to import the data:

```
import data --region=value --file=value --member=value [--invoke-callbacks=value]
```

In this scenario, data from replicated and partitioned regions is imported sequentially from a single file on the specified member.

See export data for an explanation of the difference between serial and parallel data exports.

If the data was exported from a partitioned region in parallel format, use the --parallel option to import the data:

```
import data --parallel --region=value --member=value --dir=value
  [--invoke-callbacks=value]
```

In this scenario, partitioned region data is imported simultaneously on all hosting nodes from directories local to those respective nodes.

- Parallel import applies only to partitioned regions exported in parallel mode.
- The --file and --dir options are mutually exclusive; --file can be used only for serial import, --dir can be used for both serial and parallel imports.

Import Data Parameters:

Name	Description	Default Value
--region	*Required.* Region into which data will be imported.	
--member	*Required.* Name/ID of a member that hosts the region. In a serial import, all data is imported from the specified file on the host where the member is running. In a parallel import, data from a partitioned region partially hosted on this member is imported for each partition from files on the nodes that host those partitions.	
--file	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. Cannot be specified at the same time as --dir, cannot be used with --parallel.	
--dir	Directory from which the data is to be imported. Required if --parallel is true. Cannot be specified at the same time as --file. If a given node has no data file in the specified directory, the import operation is silently skipped for that node.	

Name	Description	Default Value
--invoke-callbacks	Boolean value that, when true, invokes callbacks during the data import.	false

Example Commands:

```
import data --region=region2 --file=/mnt5/region2_20121001.gfd --member=server1
```

list

This topic describes the list command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to list existing Tanzu GemFire resources such as deployed applications, disk-stores, functions, members, servers, and regions.

- list async-event-queues

  Display a list of async event queues for all members.

- list clients

  Display a list of connected clients.

- list deployed

  Display a list of JARs that were deployed to members using the deploy command.

- list disk-stores

  List all available disk stores across the Tanzu GemFire cluster

- list durable-cqs

  List durable client CQs associated with the specified durable client id.

- list functions

  Display a list of registered functions. The default is to display functions for all members.

- list gateways

  Display the gateway senders and receivers for a member or members.

- list indexes

  Display the list of indexes created for all members.

- list initializers

  Display initializers configured in Cluster Configuration and list the initializers for all members in the given groups. Cluster configuration must be enabled to use this command.

- list jndi-binding

  List all JNDI bindings, active and configured.

- list members

  Display all or a subset of members.

- list queries

  Display a list of queries currently running in the cluster.

- list regions

Display regions of a member or members. If no parameter is specified, all regions in the cluster are listed.

- list wan-copy region

  Display the currently running wan-copy region processes and retrieve their wan-copy-ids

---

list async-event-queues

Display a list of async event queues for all members.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
list async-event-queues
```

Example Commands:

```
list async-event-queues
```

---

list clients

Display a list of connected clients and the servers to which they connect.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
list clients
```

Example Commands:

```
list clients
```

Sample Output:

```
gfsh>list clients

ClientList

        Client Name / ID          | Server Name / ID
------------------------------------- | ---------------------------------------------------
192.0.2.0(4987:loner):58922:7b3398cf | member=server2,port=53508;  member=server1,port=56806
192.0.2.0(5065:loner):39906:a6f598cf | member=server2,port=53508;  member=server1,port=56806
```

---

list deployed

Display a list of JARs that were deployed to members using the deploy command.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
list deployed [--groups=value(,value)*]
```

Parameters, list deployed:

Name	Description
--groups	Groups of members for which deployed JARs will be displayed. If not specified, JARs for all members are displayed.

Example Commands:

```
list deployed
list deployed --groups=Group2
```

Sample Output:

```
gfsh> list deployed  --groups=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 Tanzu GemFire cluster.

The command also lists the configured disk directories and any Regions, Cache Servers, Gateways, PDX Serialization, and Async Event Queues using Disk Stores to either overflow 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 [--verbose]
```

Parameters, list disk-stores:

Name	Description
--verbose	Optional. Displays additional properties of the disk store.

Output:

Property	Description
Member Name	Name of the member hosting the disk store.
Member ID	Host and port number of the member hosting the disk store.
Disk Store Name	Name assigned to the disk store.
Disk Store ID	Unique identifier assigned to the disk store.
Disk Directory	Absolute path of the folder where the disk store files are located./td>
Total Disk Space	Total disk space of the disk partition where the absolute path of the disk store directory is located. File.getTotalSpace() is used to obtain this value.

Property	Description
Total Usable Space	Total usable disk space in the disk partition where the absolute path of the disk store directory is located File.getUsableSpace() is used to obtain this value.

Example Commands:

```
list disk-stores
list disk-stores --verbose
```

Sample Output:

```
gfsh> list disk-stores

 Member Name   |                Member Id              | Disk Store Name |           Disk Store ID
-------------- | -------------------------------------------| --------------- | -------------------------------------
consumerServer | 192.0.2.0(consumerServer:13825)<v5>:3545  | consumerData    | 4029af26-fd82-4997-bd6c-33382cdbb5e9
consumerServer | 192.0.2.0(consumerServer:13825)<v5>:3545  | observerData    | 7e0316ad-963c-49b0-9b01-8f59b8d9e29e
producerServer | 192.0.2.0(producerServer:13826)<v3>:53764 | producerData    | 4670e4eb-1c50-4465-b418-08ede3d5dbed
```

```
gfsh> list disk-stores --verbose

Member Name |              Member Id              | Disk Store Name | Disk Store ID
----------- | ------------------------------------ | --------------- | -------------------------------------
server-1    | 192.168.50.8(server-1:67564)<v1>:41977 | diskstore       | 6cc9f51f-5ccf-42cf-ba60-ec8ccdbb83c2
server-2    | 192.168.50.8(server-2:67565)<v2>:45208 | diskstore       | c72202f0-8496-438b-bf88-bc7e05db1e83

Member Name | Disk Store Name |      Property      | Value
----------- | --------------- | ------------------ | ------------------------------------------------------------------------------------------------------------------------------------------------------
---------
server-1    | diskstore       | Disk Directory     | /test-worker-000001/dunit/vm1/diskDirectory
            |                 | Total Disk Space   | 926.352 GiB
            |                 | Total Usable Space | 499.023 GiB
server-2    | diskstore       | Disk Directory     | /test-worker-000001/dunit/vm2/diskDirectory
            |                 | Total Disk Space   | 926.352 GiB
            |                 | Total Usable Space | 499.023 GiB
```

Error Messages:

```
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
 [--members=value(,value)*] [--groups=value(,value)*]
```

Parameters, list durable-cqs:

Name	Description
--durable-client-id	*Required.* The ID used to identify the durable client.
--members	Name or Id of the members for which the durable client is registered and durable CQs will be displayed.

Name	Description
--groups	Groups 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:

```
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] [--groups=value(,value)*]
[--members=value(,value)*]
```

Parameters, list functions:

Name	Description
--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 “.*” rather than “*”.
--groups	Groups of members for which functions will be displayed. Use a comma separated list for multiple groups.
--members	Name or ID of the members for which functions will be displayed. Use a comma separated list for multiple members.

Example Commands:

```
list functions
list functions --matches=reconcile.*
```

Sample Output:

```
gfsh> list functions

 Member   |          Function
--------- | --------------------------
camelot   | loadDataFromExternalSource
camelot   | reconcileWeeklyExpenses
excalibur | loadDataFromExternalSource
excalibur | reconcileDailyExpenses
```

Example of list functions with a “matches” filter:

```
gfsh> list functions --matches=reconcile.*

 Member   |        Function
--------- | -----------------------
camelot   | reconcileWeeklyExpenses
excalibur | reconcileDailyExpenses
```

Error Messages:

```
No Functions Found
```

---

list gateways

Display 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 [--members=value(,value)*] [--groups=value(,value)*] [--senders-only | --receivers-only]
```

Parameters, list gateways:

Name	Description
--members	Members whose gateways senders and receiver display.
--groups	Groups of members for which Gateway Senders and Receivers will be displayed. Use a comma separated list for multiple groups.
--senders-only	List only gateway senders. This parameter cannot be used together with --receivers-only.
--receivers-only	List only gateway receivers. This parameter cannot be used together with --senders-only.

Example Commands:

```
list gateways
list gateways --senders-only
list gateways --receivers-only
```

Sample Output:

```
gfsh>list gateways

GatewaySender Section

GatewaySender Id |            Member        | Remote Cluster Id |  Type    | Status  | Queued Events | Receiver Location
---------------- | ------------------------- | ------------------ | -------- | ------- | ------------- | ------------------
ln               | mymac(ny-1:88641)<v2>:33491 | 2                 | Parallel | Running | 0             | mymac:5037
ln               | mymac(ny-2:88705)<v3>:29329 | 2                 | Parallel | Running | 0             | mymac:5064
ln               | mymac(ny-3:88715)<v4>:36808 | 2                 | Parallel | Running | 0             | mymac:5132
ln               | mymac(ny-4:88724)<v5>:52993 | 2                 | Parallel | Running | 0             | mymac:5324

GatewayReceiver Section

        Member            | Port | Sender Count | Senders Connected
--------------------------- | ---- | ------------ | ------------------------------------------------------------------------------------------------------------------------------------------------------
mymac(ny-1:88641)<v2>:33491 | 5057 | 9            |["mymac(ln-1:88651)<v2>:48277","mymac(ln-4:88681)<v5>:42784","mymac(ln
-3:88672)<v4>:43675","mymac(ln-2:88662)<v3>:12796"]
mymac(ny-2:88705)<v3>:29329 | 5082 | 4            |["mymac(ln-1:88651)<v2>:48277","mymac(ln-4:88681)<v5>:42784","mymac(ln
-3:88672)<v4>:43675"]
mymac(ny-3:88715)<v4>:36808 | 5371 | 4            |["mymac(ln-1:88651)<v2>:48277","mymac(ln-4:88681)<v5>:42784","mymac(ln
-3:88672)<v4>:43675"]
mymac(ny-4:88724)<v5>:52993 | 5247 | 3            |["mymac(ln-1:88651)<v2>:48277","mymac(ln-4:88681)<v5>:42784","mymac(ln
-3:88672)<v4>:43675"]
```

---

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)?]
```

Parameters, list indexes:

Name	Description	Default Value
--with-stats	Specifies whether statistics should also be displayed.	false

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 | 192.0.2.0(consumerServer:13873):6317      | /consumers  | cidIdx   | KEY   | id                 | /consumers
consumerServer | 192.0.2.0(consumerServer:13873):6317      | /consumers  | cnameIdx | RANGE | name               | /consumers
producerServer | 192.0.2.0(producerServer:13874):19198     | /producers  | pidIdx   | RANGE | id                 | /producers
```

Example of ‘list indexes’ with stats printed:

```
gfsh>list indexes --with-stats

Member Name  | Member ID | Region Path |   Name   | Type  | Indexed Expression | From Clause | Uses | Updates | Update Tim
e | Keys | Values
------------ | --------- | ----------- | -------- | ----- | ------------------ | ----------- | ---- | ------- | ----------- | ---- | ------
cs...        | 192...    | /consumers  | cidIdx   | KEY   | id                 | /consumers  | 2512 | 0       | 0           | 5020 | 5020
cs...        | 192...    | /consumers  | cnameIdx | RANGE | name               | /consumers  | 0    | 5020    | 421224000   | 0    | 5020
ps...        | 192...    | /producers  | pidIdx   | RANGE | id                 | /producers  | 0    | 5031    | 497872000   | 5031 | 5031
```

Error Messages:

```
No Indexes Found
```

---

list initializers

Display initializers configured in Cluster Configuration and list the initializers for all members in the given groups. Cluster configuration must be enabled to use this command.

Syntax

```
list initializers [--groups=value(,value)*]
```

Parameters, list initializer:

Name	Description
--groups	Groups for which to show configured initializers. If no group is specified, all initializers are displayed.

Example Commands:

```
list initializers --groups=group1,group2
list initializers
```

Example Output:

```
gfsh> list initializers

Configured Initializers

Group    |          Class Name
---------|--------------------------------
cluster  | TestSystemMembershipListener
MyGroup | TestSystemMembershipListener2


Initializer Applied to Member

Member   |          Class Name
---------| -------------------------------
server_1 | TestSystemMembershipListener
server_2 | TestSystemMembershipListener
server_3 | Not Initialized
server_4 | TestSystemMembershipListener2
server_5 | No Initializer Configured
```

---

list jndi-binding

List all JNDI bindings, active and configured. An active binding is bound to the server's JNDI context and is also listed in the cluster configuration. A configured binding is listed in the cluster configuration, but may not be active on the servers.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
list jndi-binding
```

Sample Output:

```
gfsh>list jndi-binding

Configured JNDI bindings:

Group Name | JNDI Name | JDBC Driver Class
---------- | --------- | ------------------------------------
cluster    | jndi1     | org.apache.derby.jdbc.EmbeddedDriver

Active JNDI bindings found on each member:

    Member      |        JNDI Name        | JDBC Driver Class
--------------- | ----------------------- | ----------------------------------------------------
land-gifted-gun | java:UserTransaction    | org.apache.geode.internal.jta.UserTransactionImpl
land-gifted-gun | java:TransactionManager | org.apache.geode.internal.jta.TransactionManagerImpl
```

list members

Display all or a subset of members, with the option of displaying additional information about each member.

Within the output, the membership coordinator is listed. identifies the view the member currently has. In <vN>, N will be zero or a positive interger. indicates which members are eligible to be a membership coordinator. If no value can be retrieved for a member, the field will be denoted as NA, with the exception of the redundancy-zone column. If the redundancy zone is not set or cannot be retrieved, it will be denoted by a blank space.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
list members [--group=value] [--verbose] [--sort-column=value]
```

Parameters, list members:

Name	Description
--group	Group name for which members will be displayed.
--verbose	Displays additional information about the members.
--sort-column	The column name on which the result table is sorted. If two or more values in the sorted column are the same, then the members with the same values are sorted by name. The Port column cannot be sorted because multiple ports may be present. If no sort column is provided, then the results are sorted by values in the Type column.

Column	Description
Name	The name of the member.

Column	Description
ID	The unique ID created by GemFire and assigned to the member. The member that is assigned as the cluster coordinator will include [Coordinator] in the ID. For more information about cluster coordinators, see Membership Coordinators, Lead Members, and Member Weighting.
Type	The type of the member. The type will be either Locator or Server.
Status	The current status of the member. The status will be one of the following: • Initializing: The member is starting. • Ready: The member has started successfully and is ready to connect to clients. Use the describe member command to see the member startup lifecycle.
Host	The host of the member.
Port	The port number of the member.
LoadAverage1	The average number of threads on the host machine that are in the run queue or are waiting for disk I/O over the last minutes. This value is retrieved from /proc/loadavg and helps determine if the load on the system is too high for the number of CPUs. In a multi-processor system, the load is proportional to the number of processor cores in use. For example, "100% utilization" corresponds to load average of 1.0 on a single-core system, 2.0 on a dual-core system, 4.0 on a quad-core system, etc. and means that, on average, one CPU core has been fully utilized during that period, providing insights into system activity and performance.
Max-Heap	The maximum amount of memory, in megabytes (MB), that can be allocated to the member's heap space.
Used-Heap	The current usage of heap space by the member, in megabytes (MB).
Cache-Entries	The total number of entries in the cache.
Redundancy-Zone	The name of the redundancy zone that has been configured for the member. If no redundancy zone has been configured for the member, this field is empty.
Uptime	The duration for which the member has been running continuously without any restarts.

Example Commands:

```
list members
list members --group=Group1
list members --verbose
list members --sort-column=Name
list members --verbose --sort-column=Uptime
```

Sample Output:

```
gfsh>list members

Name      | Id                                                                | Type    | Status |
--------- | ------------------------------------------------------------------| ---———----| ---———----|
locator-0 | 192.168.43.99(locator-0:16523:locator)<ec><v0>:42586 [Coordinator] | Locator | READY  |
locator-1 | 192.168.43.99(locator-1:16524:locator)<ec><v1>:42863              | Locator | READY  |
server-2  | 192.168.43.99(server-2:16525)<v2>:47137                           | Server  | READY  |
server-3  | 192.168.43.99(server-3:16526)<v3>:44209                           | Server  | READY  |
server-4  | 192.168.43.99(server-4:16611)<v4>:52532                           | Server  | READY  |
```

```
gfsh>list members --sort-column=Name
Name      | Id                                                                | Type
--------- | ------------------------------------------------------------------ | —------
locator-0 | 192.168.43.99(locator-0:16523:locator)<ec><v0>:42586 [Coordinator] | Locator
locator-1 | 192.168.43.99(locator-1:16524:locator)<ec><v1>:42863               | Locator
server-2  | 192.168.43.99(server-2:16525)<v2>:47137                            | Server
server-3  | 192.168.43.99(server-3:16526)<v3>:44209                            | Server
server-4  | 192.168.43.99(server-4:16611)<v4>:52532                            | Server
```

```
gfsh>list members --verbose
Name      |                          Id                          |  Type   | Status |     Host     | Port |  LoadAverage1  | Max-Heap  |  Used-H
eap  | Cache-Entries | Redundancy-Zone | Uptime
--------- | ------------------------------------------------------------------ | ------- | ------ | ------------ | ----- | -------------- | --------- | ------------ | ------------
-- | --------------- | ----------
locator-0 | 192.168.50.8(locator-0:67023:locator)<ec><v0>:49006 [Coordinator] | Locator | Ready  | 192.168.50.8 | 20004 | 41.48
291015625 | 473.5 MiB | 251.253 MiB | 1             |                | 21 seconds
locator-1 | 192.168.50.8(locator-1:67024:locator)<ec><v1>:59945               | Locator | Ready  | 192.168.50.8 | 20007 | 41.482910
15625 | 472.5 MiB | 201.81 MiB  | 1             |                | 21 seconds
server-2  | 192.168.50.8(server-2:67025)<v2>:57010                            | Server  | Ready  | 192.168.50.8 | 20010 | 41.4829101562
5 | 491 MiB   | 165.408 MiB | 0             | RACK-1         | 21 seconds
server-3  | 192.168.50.8(server-3:67026)<v3>:55582                            | Server  | Ready  | 192.168.50.8 | 20014 | 41.4829101562
5 | 491 MiB   | 168.577 MiB | 0             | RACK-1         | 21 seconds
server-4  | 192.168.50.8(server-4:67064)<v4>:46713                            | Server  | Ready  | 192.168.50.8 | 20018 | 41.4829101562
5 | 491 MiB   | 164.213 MiB | 0             | RACK-2         | 7 seconds
```

```
gfsh>list members --verbose --sort-column=Type
Name      |                          Id                          |  Type   | Status |     Host     | Port |  LoadAverage1  | Max-Heap  |  Used-H
eap  | Cache-Entries | Redundancy-Zone | Uptime
--------- | ------------------------------------------------------------------ | ------- | ------ | ------------ | ----- | -------------- | --------- | ------------ | ------------
-- | --------------- | ----------
locator-0 | 192.168.50.8(locator-0:67023:locator)<ec><v0>:49006 [Coordinator] | Locator | Ready  | 192.168.50.8 | 20004 | 41.48
291015625 | 473.5 MiB | 251.253 MiB | 1             |                | 21 seconds
locator-1 | 192.168.50.8(locator-1:67024:locator)<ec><v1>:59945               | Locator | Ready  | 192.168.50.8 | 20007 | 41.482910
15625 | 472.5 MiB | 201.81 MiB  | 1             |                | 21 seconds
server-2  | 192.168.50.8(server-2:67025)<v2>:57010                            | Server  | Ready  | 192.168.50.8 | 20010 | 41.4829101562
5 | 491 MiB   | 165.408 MiB | 0             | RACK-1         | 21 seconds
server-3  | 192.168.50.8(server-3:67026)<v3>:55582                            | Server  | Ready  | 192.168.50.8 | 20014 | 41.4829101562
5 | 491 MiB   | 168.577 MiB | 0             | RACK-1         | 21 seconds
server-4  | 192.168.50.8(server-4:67064)<v4>:46713                            | Server  | Ready  | 192.168.50.8 | 20018 | 41.4829101562
5 | 491 MiB   | 164.213 MiB | 0             | RACK-2         | 7 seconds
```

---

list queries

Display a list of queries currently running in the cluster.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
list queries [--limit=value] [--(member|members)=value(,value)*]
```

Parameters, list queries:

Name	Description
--limit	Maximum number of queries to display.
--member or --members	Name or ID of the members for which queries will be displayed. If this is not specified then queries from all members will be displayed.

Example Commands:

```
list queries
list queries --members=server1,server3
list queries --limit=5
list queries --limit=5 --members=server1,server3
```

Sample Output:

```
gfsh>list queries

Query ID                          | Elapsed Time | Query String
------------------------------------ | -------------| --------------------
93b228ea-717a-4ec3-8208-ac1e3faa1c34 | 01:25:15.545 | SELECT * FROM /region
faf13b3b-0da1-4b75-885f-2dc1a010f080 | 00:34:14:222 | SELECT * FROM /region
[Time format is hh:mm:ss.sss.]
```

Elapsed Time shown is the maximum elapsed time in seconds from all members where the query is being executed, or for the members as specified by the --member or --members arguments.

---

list regions

Display regions of a member or members, with the option of displaying additional information about each region. If no parameter is specified, all regions in the cluster are listed.

Syntax:

```
list regions [--groups=value(,value)*] [--members=value(,value)*] [--verbose] [--sort-column=value]
```

Parameters, list regions:

Name	Description
--groups	Groups of members for which regions will be displayed.
--members	Name or ID of the members for which regions will be displayed.
--verbose	Displays additional information about the regions.
--sort-column	The column name on which the result table is sorted. If two or more values in the sorted column are the same, then the regions with the same values are sorted by name.

Column	Description
Name	The name of the region.
Status	The current status of the region. The status will be one of the following: • Initializing Indexes • Ready
Data-Policy	How local cache will handle the data for the region.
Number-of-Entries	The number of entries present in the cache for the region.
Size-of-Values	The size of the values in the cache for the region.
Disk-Store	The name of the disk store associated with the region.
Redundancy	Redundancy state of the entries in the region, if satisfied or not.

Example Commands:

```
list regions
list regions --groups=G1
list regions --members=member1
list regions --verbose
```

Sample Output:

```
gfsh>list regions

List of regions
---------------
region1
region2
```

```
gfsh>list regions --verbose

             Name                | Status | Data-Policy | Number-of-Entries | Size-of-Values | Disk-Store | Redundancy
------------------------------------ | ------ | ----------- | ----------------- | -------------- | ---------- | -------------
regionWithSubregions               | Ready  | PRELOADED   | 1                 | 2.109 KiB      |            | Not Available
regionWithSubregions/localSubregion | Ready  | PRELOADED   | 1000              | 2.06 MiB       |            | Not Available
```

```
gfsh>list regions --verbose --sort-column=Size-of-Values

             Name                | Status | Data-Policy | Number-of-Entries | Size-of-Values | Disk-Store | Redundancy
------------------------------------ | ------ | ----------- | ----------------- | -------------- | ---------- | -------------
regionWithSubregions/localSubregion | Ready  | PRELOADED   | 1000              | 2.06 MiB       |            | Not Available
regionWithSubregions               | Ready  | PRELOADED   | 1                 | 2.109 KiB      |            | Not Available
```

---

list wan-copy region

Use this command to see currently running wan-copy region processes and retrieve their wan-copy-ids. For more information, see wan-copy region.

Syntax

```
list wan-copy region.
```

Parameters

This command has no parameters and provides the following information in a table format:

```
Id: The wan-copy-id
Regions: The original region expression for each command
Status: The current status of each wan-copy-id

Create Time: The date the request was created.
Elapsed Time: The duration each process has been running or the total run time if it has finished
```

Example 1

```
list wan-copy region

                Id                | Regions           | Status   | Create Time              | Elapsed Time
-------------------------------------|--------------------|----------|--------------------------|---------------------
c412ed7a-23e9-4533-8ace-71242ddfa78d | region1*          | FINISHED | May 21 09:37:39 PDT 2024 | 2 hrs 45 min 33 sec
ce83240d-e1dd-49b3-ad9b-eed2d3c90d5b | consumer1,consum...| FAILED   | May 22 08:30:31 PDT 2024 | 0 hrs 23 min 15 sec
2bdb59a2-e177-4d86-b320-b3c50087f2fb | myRegion*         | CANCELED | May 22 08:30:32 PDT 2024 | 0 hrs 0 min 15 sec
```

load-balance gateway-sender

This topic describes the load-balance gateway-sender command in gfsh, the VMware Tanzu GemFire command-line interface.

This command causes the specified gateway sender to close its current connections and reconnect to remote gateway receivers in a more balanced fashion.

Use this command to load balance connections between gateway senders to receivers. For example, when you add a new gateway receiver node at a remote site, execute this command so that the new gateway receiver can pick up connections from the specified gateway sender. Invoking this command redistributes a sender's connections more evenly among all the gateway receivers.

> This command has no effect on ping connections.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
load-balance gateway-sender --id=value
```

Name	Description
--id	*Required.* ID of the Gateway Sender.

Table 1. Load-Balance Gateway-Sender Parameters

Example Commands:

```
load-balance gateway-sender --id=sender1-LN
```

Sample Output:

```
load-balance gateway-sender --id=ny

                Member            | Result | Message
 ---------------------------------- | ------ |-----------------------------------------------------------------------------------------
boglesbymac(ln-1:88651)<v2>:48277 | OK     | GatewaySender ny is rebalanced on member boglesbymac(ln-1:88651)<v2>:482
77
boglesbymac(ln-4:88681)<v5>:42784 | OK     | GatewaySender ny is rebalanced on member boglesbymac(ln-4:88681)<v5>:427
84
boglesbymac(ln-3:88672)<v4>:43675 | OK     | GatewaySender ny is rebalanced on member boglesbymac(ln-3:88672)<v4>:436
75
boglesbymac(ln-2:88662)<v3>:12796 | OK     | GatewaySender ny is rebalanced on member boglesbymac(ln-2:88662)<v3>:127
96
```

locate entry

This topic describes the locate entry command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to 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]
```

Name	Description	Default 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.	java.lang.String
--value-class	Fully qualified class name of the value’s type.	java.lang.String
--recursive	Whether to traverse regions and subregions recursively.	false

Table 1. Locate Entry Parameters

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

This topic describes the netstat command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to report network usage information and statistics for a specified member through use of the “netstat” operating system command.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
netstat [--members=value(,value)*] [--group=value] [--file=value]
[--with-lsof(=value)?]
```

Name	Description	Default Value
--members	Name or ID of the members 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 is not already part of the specified name.	
--with-lsof	Specifies whether lsof (list open files) command output should also be displayed. Not applicable for Microsoft Windows hosts.	false

Table 1. Netstat Parameters

Example Commands:

```
netstat
netstat --members=server1
netstat --members=server1 --file=server1_netstat.txt
```

Sample Output:

```
gfsh>netstat
#####################################
Host: GemFireTest
OS: Windows XP 5.1 x86
Members:
 server2, locator1, server1
#####################################

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    GemFireTest:epmap    GemFireTest:0        LISTENING
  TCP    GemFireTest:microsoft-ds  GemFireTest:0        LISTENING
  TCP    GemFireTest:1034     GemFireTest:0        LISTENING
  TCP    GemFireTest:1069     GemFireTest:0        LISTENING
  TCP    GemFireTest:1099     GemFireTest:0        LISTENING
  TCP    GemFireTest:1134     GemFireTest:0        LISTENING
  TCP    GemFireTest:3389     GemFireTest:0        LISTENING
  TCP    GemFireTest:8080     GemFireTest:0        LISTENING
  TCP    GemFireTest:8081     GemFireTest:0        LISTENING
  TCP    GemFireTest:10334    GemFireTest:0        LISTENING
  TCP    GemFireTest:40404    GemFireTest:0        LISTENING
  TCP    GemFireTest:40405    GemFireTest:0        LISTENING
  TCP    GemFireTest:1025     GemFireTest:0        LISTENING
  TCP    GemFireTest:5152     GemFireTest:0        LISTENING
  TCP    GemFireTest:netbios-ssn  GemFireTest:0        LISTENING
  TCP    GemFireTest:1035     GemFireTest:0        LISTENING
  TCP    GemFireTest:1035     GemFireTest:1081     ESTABLISHED
  TCP    GemFireTest:1035     GemFireTest:1086     ESTABLISHED
  TCP    GemFireTest:1035     GemFireTest:1147     ESTABLISHED
  TCP    GemFireTest:1035     GemFireTest:1156     ESTABLISHED
  TCP    GemFireTest:1046     GemFireTest:1099     ESTABLISHED
  TCP    GemFireTest:1049     osdc-proxy-vip.vmware.com:http  CLOSE_WA
  TCP    GemFireTest:1050     osdc-proxy-vip.vmware.com:3128  CLOSE_WA
  TCP    GemFireTest:1071     GemFireTest:0        LISTENING
  TCP    GemFireTest:1071     GemFireTest:1077     ESTABLISHED
  TCP    GemFireTest:1071     GemFireTest:1150     ESTABLISHED
  TCP    GemFireTest:1071     GemFireTest:1157     ESTABLISHED
  TCP    GemFireTest:1077     GemFireTest:1071     ESTABLISHED
  TCP    GemFireTest:1078     GemFireTest:24400    ESTABLISHED
  TCP    GemFireTest:1081     GemFireTest:1035     ESTABLISHED
  TCP    GemFireTest:1086     GemFireTest:1035     ESTABLISHED
  TCP    GemFireTest:1099     GemFireTest:1046     ESTABLISHED
  TCP    GemFireTest:1136     GemFireTest:0        LISTENING
  TCP    GemFireTest:1136     GemFireTest:1143     ESTABLISHED
  TCP    GemFireTest:1136     GemFireTest:1151     ESTABLISHED
  TCP    GemFireTest:1136     GemFireTest:1201     ESTABLISHED
  TCP    GemFireTest:1141     GemFireTest:4247     ESTABLISHED
  TCP    GemFireTest:1142     GemFireTest:48640    ESTABLISHED
  TCP    GemFireTest:1143     GemFireTest:1136     ESTABLISHED
  TCP    GemFireTest:1147     GemFireTest:1035     ESTABLISHED
  TCP    GemFireTest:1150     GemFireTest:1071     ESTABLISHED
  TCP    GemFireTest:1151     GemFireTest:1136     ESTABLISHED
  TCP    GemFireTest:1156     GemFireTest:1035     ESTABLISHED
  TCP    GemFireTest:1157     GemFireTest:1071     ESTABLISHED
  TCP    GemFireTest:1201     GemFireTest:1136     ESTABLISHED
  TCP    GemFireTest:1349     GemFireTest:10334    TIME_WAIT
  TCP    GemFireTest:1350     GemFireTest:10334    TIME_WAIT
  TCP    GemFireTest:1351     GemFireTest:10334    TIME_WAIT
  TCP    GemFireTest:1352     GemFireTest:10334    TIME_WAIT
  TCP    GemFireTest:1353     GemFireTest:10334    TIME_WAIT
  TCP    GemFireTest:1354     GemFireTest:10334    TIME_WAIT
  TCP    GemFireTest:4247     GemFireTest:0        LISTENING
```

```
  TCP    GemFireTest:4247     GemFireTest:1141     ESTABLISHED
  TCP    GemFireTest:24400    GemFireTest:0        LISTENING
  TCP    GemFireTest:24400    GemFireTest:1078     ESTABLISHED
  TCP    GemFireTest:48640    GemFireTest:0        LISTENING
  TCP    GemFireTest:48640    GemFireTest:1142     ESTABLISHED
  UDP    GemFireTest:microsoft-ds  *:*
  UDP    GemFireTest:isakmp   *:*
  UDP    GemFireTest:4500     *:*
  UDP    GemFireTest:ntp      *:*
  UDP    GemFireTest:1900     *:*
  UDP    GemFireTest:ntp      *:*
  UDP    GemFireTest:netbios-ns  *:*
  UDP    GemFireTest:netbios-dgm  *:*
  UDP    GemFireTest:1900     *:*
  UDP    GemFireTest:32270    *:*
  UDP    GemFireTest:42838    *:*
  UDP    GemFireTest:47727    *:*
```

pause gateway-sender

This topic describes the pause gateway-sender command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to pause a gateway sender on a member or members. For details about pausing gateway senders, see Pausing Gateway Senders.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
pause gateway-sender --id=value [--groups=value(,value)*]
[--members=value(,value)*]
```

Name	Description
--id	*Required.* ID of the gateway sender.
--groups	Groups of members on which to pause the gateway sender.
--members	Name or ID of the members on which to pause the gateway sender.

Table 1. Pause Gateway-Sender Parameters

Example Commands:

```
pause gateway-sender --id=sender1
```

pdx rename

This topic describes the pdx rename command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to rename PDX types in an offline disk store.

Any PDX types that are renamed will be listed in the output. If no renames exist or the disk-store is online, then this command will fail.

Availability: Offline.

Syntax:

```
pdx rename --old=value --new=value --disk-store=value --disk-dirs=value(,value)*
```

Name	Description
--old	*Required.* If a PDX type's fully qualified class name has a word that matches this text then it will be renamed. Words are delimited by '.' and '$'.
--new	*Required.* The text to replace the word that matched old.
--disk-store	*Required.* Name of the disk store to operate on.
--disk-dirs	*Required.* Directories which contain the disk store files.

Example Commands:

Change all packages that start with "com.gemstone" to "com.vmware":

```
gfsh>pdx rename --old=com.gemstone --new=com.vmware --disk-store=ds1 --disk-dirs=/diskDir1
```

Change a class named "MyClassName" to "YourClassName":

```
gfsh>pdx rename --old=MyClassName --new=YourClassName --disk-store=ds1 --disk-dirs=/diskDir1
```

Change the FQCN "com.target.app1.OldClass" to "com.target.app2.NewClass":

```
gfsh>pdx rename --old=com.target.app1.OldClass --new=com.target.app2.NewClass --disk-store=ds1 --disk-dirs=/diskDir1
```

Sample Output:

```
gfsh>pdx rename --old=PortfolioPdx --new=StockPdx --disk-store=DiskStore1 --disk-dirs=/DiskDir1
Successfully renamed pdx types:
  com.app.data.StockPdx: id=2
  com.app.data.StockPdx$Day.Monday
```

Error Messages:

If no types match, you may receive the following error message:

```
gfsh>pdx rename --old=gemstone --new=vmware --disk-store=DiskStore1 --disk-dirs=/DiskDir1
Could not process command due to GemFire error. No Pdx types found to rename.
```

If the disk-store where the PDX types are stored is online, you will receive the following error message:

```
gfsh>pdx rename --old=StockPdx --new=PortfolioPdx --disk-store=DiskStore1 --disk-dirs=/DiskDir1
Could not process command due to GemFire error. Error renaming pdx types :
GemFireCache[id = 484629896; isClosing = false; isShutDownAll = false;
closingGatewayHubsByShutdownAll = false; created = Wed Jan 07 10:29:45 PST 2015;
server = false; copyOnRead = false; lockLease = 120; lockTimeout = 60]: An open cache
already exists.
```

put

This topic describes the put command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to add or update a region entry.

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] [--if-not-exists(=value)]
```

Name	Description	Default Value
--key	*Required.* String or JSON text from which to create the key. For example: “James”, “100L” and “('id':'l34s')”.	
--value	*Required.* String or JSON text from which to create the value. For example: “James”, “100L” and “('id':'l34s')”.	
--region	*Required.* Region into which the entry will be put.	
--key-class	Fully qualified class name of the key’s type.	java.lang.String
--value-class	Fully qualified class name of the value’s type. The type may not be an internal PDX class.	java.lang.String
--value-class	Fully qualified class name of the value’s type. The type may not be an internal PDX class.	java.lang.String
--if-not-exists	Skip the put operation when an entry with the same key already exists.	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

This topic describes the query command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to run queries against Tanzu GemFire regions.

If a limit restricting the result size is not set in the query, then a default limit of the gfsh environment variable APP_FETCH_SIZE is applied. For more information about APP_FETCH_SIZE, see Useful gfsh Shell

Variables.

When using the query command, surround the OQL query with single quotation marks. The OQL query must be a SELECT expression that references at least one region.

> Do not execute this command from gfsh if the objects being queried contain cyclic references. Do not specify a member using --member when you run queries on replicate regions because you do not need to or you might point to a non-data member and get an error message.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
query --query=value [--file=path/to/results/file] [--member=member-name]
```

Name	Description
--query	*Required*. The OQL string. This must be a SELECT expression that references at least one region.
--file	When specified, all query results are written to the specified file. An error is issued if the file already exists.
--member	Name or ID of the member on which to execute the query. You can find the name or ID in the output of the gfsh describe region command, for example.

Sample Output:

```
gfsh>query --query='SELECT * FROM /region2'

Result     : true
startCount : 0
endCount   : 20
Rows       : 1

Result
------------------
('Hello World!!')

NEXT_STEP_NAME : END
```

rebalance

This topic describes the rebalance command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to 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)?]
```

Name	Description	Default Value
--include-region	Partitioned Region paths to be included for rebalance operation. Includes take precedence over excludes.	

Name	Description	Default Value
--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.	-1 (wait for rebalancing to complete
--simulate	Whether to only simulate rebalancing. The –time-out parameter is not available when simulating.	false

Table 1. Rebalance Parameters

Example Commands:

```
rebalance --include-region=/region3 --simulate=true
```

Sample Output:

```
                                    Rebalanced Stats                                    | Value
------------------------------------------------------------------------------------------ | -----
Total bytes in all redundant bucket copies created during this rebalance                | 0
Total time (in milliseconds) spent creating redundant bucket copies during this rebalance      | 0
Total number of redundant copies created during this rebalance                          | 0
Total bytes in buckets moved during this rebalance                                       | 0
Total time (in milliseconds) spent moving buckets during this rebalance                  | 0
Total number of buckets moved during this rebalance                                      | 0
Total time (in milliseconds) spent switching the primary state of buckets during this rebalance | 0
Total primaries transferred during this rebalance                                        | 0
Total time (in milliseconds) for this rebalance                                          | 0
Total number of members in system on which rebalance is executed                         | 2
```

remove

This topic describes the remove command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to 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]
```

Name	Description	Default 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.	key constraint for the current region or String
--region	*Required.* Region from which to remove the entry.	
--all	A boolean value that, when true, clears the region by removing all entries. This option is not available for partitioned regions.	false

Example Commands:

```
gfsh>remove --region=/region1 --key=('id': '133abg134')
gfsh>remove --region=/region1 --key=('id': '133abg134') --key-class=data.ProfileKey
gfsh>remove --region=/region1 --all=true
```

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"

"Option --all is not supported on partitioned region"
```

restore redundancy

This topic describes the restore redundancy command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to restore redundancy to partitioned regions and optionally reassign which members host the primary copies.

The default is for all partitioned regions to have redundancy restored and to reassign primary hosts. If any region that would have redundancy restored is a member of a colocated group, all other regions that are part of that group will also have their redundancy restored. This behavior takes precedence over any included or excluded regions specified as part of the command. For more information, see Data Colocation Between Regions

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
restore redundancy [--include-region=value(,value)*] [--exclude-region=value(,value)*] [--reassign-primaries(=value)]
```

Name	Description	Default Value
--include-region	Partitioned Region paths to be included for restore redundancy operation. Includes take precedence over excludes.	
--exclude-region	Partitioned Region paths to be excluded for restore redundancy operation.	
--reassign-primaries	If false, this operation will not attempt to reassign which members host primary buckets.	true

Example Commands:

```
restore redundancy
restore redundancy --include-region=/region3,/region2 --exclude-region=/region1
```

Sample Output:

```
restore redundancy --include-region=/region3,/region2 --exclude-region=/region1

Number of regions with zero redundant copies = 0
Number of regions with partially satisfied redundancy = 0
Number of regions with fully satisfied redundancy = 2

Redundancy is fully satisfied for regions:
  region3 redundancy status: SATISFIED. Desired redundancy is 2 and actual redundancy is 2.
  region2 redundancy status: SATISFIED. Desired redundancy is 1 and actual redundancy is 1.

Total primary transfers completed = 224
Total primary transfer time (ms) = 4134
```

resume

This topic describes the resume command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to modify an existing Tanzu GemFire resource.

- resume async-event-queue-dispatcher

  Resume dispatching of events from the queue to the listeners for a specified asynchronous event queue which is currently in a paused state.

- resume gateway-sender

  Resume specified gateway sender that is currently in a paused state.

resume async-event-queue-dispatcher

Resume dispatching of events on a specified asynchronous event queue which is currently in a paused state.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
resume async-event-queue-dispatcher --id=value [--groups=value(,value)*]
[--members=value(,value)*]
```

Name	Description
--id	*Required.* ID of the Asynchronous Event Queue.
--groups	Groups of members on which to resume event dispatching for this queue.
--members	Name or ID of the members on which to resume event dispatching for this queue.

Example Commands:

```
resume async-event-queue-dispatcher --id=AEQ1 --groups=Group1
```

resume gateway-sender

Resume a gateway sender which is currently in a paused sate.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
resume gateway-sender --id=value [--groups=value(,value)*]
[--members=value(,value)*]
```

Name	Description
--id	*Required.* ID of the Gateway Sender.
--groups	Groups of members on which to resume the Gateway Sender.
--members	Name/Id of the members on which to resume the Gateway Sender.

Example Commands:

```
resume gateway-sender --id=sender1-LN --groups=LN-Group1
```

revoke missing-disk-store

This topic describes the revoke missing-disk-store command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to instruct the members of a cluster to stop waiting for a disk store to be available.

Only revoke a disk store if its files are lost because it will no longer be recoverable after revoking is initiated. Use the show missing-disk-store command to view 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
```

Name	Description
--id	*Required.* ID of the missing disk store to be revoked.

Table 1. Revoke Missing-Disk-Store Parameters

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

This topic describes the run command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to execute a set of gfsh commands.

When using this, commands that 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)?]
```

Table 1. Parameters

Name	Description	Default Value
--file	*Required.* Path of the file scripted with commands that gfsh recognizes. Path should be relative or absolute.	
--quiet	Specifies whether to show command output. Note: All commands in a script are run non-interactively when the run command is used. This option does not change that functionality.	false
--continue-on-error	Specifies whether further execution of the script should continue if there is an error while executing one of the commands fails.	false

Table 1. Run Parameters

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" does not exist.
```

set variable

This topic describes the set variable command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to set variables in the gfsh environment.

Set gfsh variables that can be used by commands.

You can use the gfsh echo 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 for a description of preset environment variables.

Availability: Online or offline.

Syntax:

```
set variable --name=value --value=value
```

Name	Description
--name	*Required.* The name to be assigned for the variable. The name must only be composed of letters, digits, and the underscore character (_); the name may not start with a digit.
--value	*Required.* Value that the variable will be set to.

Example Commands:

```
set variable --name=FOO --value="foo"
set variable --name=BAR --value="bar"
```

Sample Output:

```
gfsh>set variable --name=BAR --value="bar"
Value for variable BAR is now: "bar".
```

sh

This topic describes the sh command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to execute operating system commands.

This command executes operating system (OS) commands. Use “&” to return to the gfsh prompt immediately.

> Commands that pass output to another shell command are not supported.

Syntax:

```
sh command [--use-console(=value)?]
```

Name	Description	Default Value
--use-console	Set this parameter on UNIX systems for applications which need a handle to the console. Adds “</dev/tty >/dev/tty” to the specified command.	false

Table 1. Sh Parameters

Example Commands:

```
gfsh>sh ls -al
total 80
drwxrwxr-x. 10 username username  4096 Sep  3 15:10 .
drwxrwxr-x.  4 username username  4096 Sep  3 14:58 ..
drwx------.  2 username username  4096 Sep  3 15:09 bin
drwx------.  2 username username  4096 Sep  3 15:09 defaultConfigs
drwx------.  3 username username  4096 Sep  3 15:09 docs
drwx------.  2 username username  4096 Sep  3 15:09 dtd
-rwx------.  1 username username 31830 Sep  3 15:09 EULA.txt
drwx------.  2 username username  4096 Sep  3 15:09 lib
drwx------.  6 username username  4096 Sep  3 15:09 SampleCode
drwx------.  4 username username  4096 Sep  3 15:09 templates
drwx------.  5 username username  4096 Sep  3 15:09 tools
```

show

This topic describes the show command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to display deadlocks, logs, metrics, missing disk-stores, and statistics.

- show dead-locks

  Display any deadlocks in the cluster.

- show log

  Display the log for a member.

- show metrics

  Display or export metrics for the entire cluster, a member, or a region.

- show missing-disk-stores

  Display a summary of the disk stores that are currently missing from the cluster.

- show statistics

  Display one or more point-in-time statistics for one or more members.

- show subscription-queue-size

  Shows the number of events in the subscription queue.

show dead-locks

Display any deadlocks in the cluster.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
show dead-locks --file=value
```

Name	Description
--file	*Required.* Name of the file to which dependencies between members will be written.

Table 1. Show Dead-Locks Parameters

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]
```

Name	Description	Default 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.	0

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)-192.0.2.0> tid=0x57] (tid=12 msgId=4) Parent region for
"region1/subregion1" doesnt exi
st.

[info 2012/09/25 14:04:51.372 PDT locator1 <RMI TCP Connection(14)-192.0.2.0> tid=0x57] (tid=12 msgId=5) Error occurred wh
ile executing "create region --n
ame=region1/subregion1 --type=LOCAL".

[info 2012/09/25 15:14:34.314 PDT locator1 <RMI TCP Connection(159)-192.0.2.0> tid=0x68] (tid=13 msgId=6) Error occurred w
hile executing "show log --membe
r=server1 --lines=5".
```

show metrics

Display or export metrics for the entire cluster, a member, or a region.

When no command line arguments are given, metrics under the categories of cluster, cache, diskstore, and query are displayed.

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)*]
```

Table 3. Show Metrics Parameters

Name	Description

Name	Description
--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: • region specified: cluster, region, partition, diskstore, callback, eviction • member specified: member, jvm, region, serialization, communication, function, transaction, diskstore, lock, eviction, distribution, offheap • member and region specified: region, partition, diskstore, callback, eviction

Table 3. Show Metrics Parameters

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     | queryRequestRate      | 0
```

show missing-disk-stores

Display a summary of the disk stores that are currently missing from the cluster.

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:

```
gfsh> show missing-disk-stores

          Disk Store ID             |   Host    |              Directory
------------------------------------ | --------- | --------------------------------------
60399215-532b-406f-b81f-9b5bd8d1b55a | excalibur | /usr/local/gemfire/deploy/disk_store1
```

show statistics

Shows statistics for one or more members of the cluster without VSD.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
show statistics --statistics=value(,value)* [--member=value(,value)*]
```

Name	Description	Default Value
--statistics	*Required.* One or more statistic to display formatted as type.name.statistic. These components can be found in VSD. Optionally, a statistic column can be renamed by following it with 'AS' followed by a column name.	

Name	Description	Default Value
--member	Name/ID of the member whose log file will be displayed. If not specified, statistics from all members will be shown.	

Example Commands:

```
show statistics --statistics=VMStats.vmStats.cpus,CachePerfStats.cachePerfStats.queryExecutions --members=server-1,server-2
```

With optional column name:

```
show statistics --statistics="VMStats.vmStats.cpus AS CPUs,CachePerfStats.cachePerfStats.queryExecutions AS Executions" --members=server-1,server-2
```

Sample Output:

```
gfsh>show statistics --statistics=VMStats.vmStats.cpus,CachePerfStats.cachePerfStats.queryExecutions --members=server-1,server-2
Member Name | VMStats.vmStats.cpus | CachePerfStats.cachePerfStats.queryExecutions
----------- | -------------------- | ---------------------------------------------
server-1    | 10                   | 0
server-2    | 10                   | 0
```

With optional column name:

```
gfsh>show statistics --statistics="VMStats.vmStats.cpus AS CPUs,CachePerfStats.cachePerfStats.queryExecutions AS Executions" --members=server-1,server-2
Member Name | CPUs | Executions
----------- | ---- | ----------
server-1    | 10   | 0
server-2    | 10   | 0
```

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
 [--members=value(,value)*] [--groups=value(,value)*]
```

Name	Description
--durable-client-id	*Required*. The ID used to identify the durable client.
--durable-cq-name	The name that identifies the CQ.
--members	Name/Id of the members for which subscription queue events are to be counted.
--groups	Groups of members for which subscription queue events are to be counted.

Table 4. Show Subscription-Queue-Size Parameters

Example Commands:

```
show subscription-queue-size --durable-client-id=client1
```

Sample Output:

```
gfsh>show subscription-queue-size --durable-client-Id=client1
member  | subcription-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

This topic describes the shutdown command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to stop all members.

This command asks all the members that have a cache to close the cache and disconnect from the system. If the --include-locators parameter is specified, the command shuts down any running locators one by one. The timeout parameter allows you to specify that the system should be shutdown forcibly after the time has exceeded.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
shutdown [--time-out=value] [--include-locators=value]
```

Name	Description	Default Value
--time-out	Time to wait (in seconds) for a graceful shutdown. Should be at least 10 seconds. The default value if not specified is 10.	10
--include-locators	To shutdown locators, specify this option as true.	false

Table 1. Shutdown Parameters

Example Commands:

```
shutdown
shutdown --time-out=15
shutdown --include-locators=true
```

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
Shutdown is triggered
```

sleep

This topic describes the sleep command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to delay gfsh command execution.

Delay for a specified amount of time in seconds. Floating point values are allowed.

Availability: Online of offline.

Syntax:

```
sleep [--time=value]
```

Name	Description	Default Value
--time	Number of seconds to sleep.	3

Table 1. Sleep Parameters

Example Commands:

```
sleep
sleep --time=60
```

Sample Output:

```
gfsh>sleep --time=60

gfsh>
```

start

This topic describes the start command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to start servers, locators, gateway senders and gateway receivers, and monitoring tools.

- start gateway-receiver

  Start the gateway receiver on a given member or group of members.

- start gateway-sender

  Start the gateway sender on a member or members.

- start jconsole

  Start the JDK JConsole monitoring application in a separate process.

- start jvisualvm

  Start the JDK's Java VisualVM monitoring application in a separate process.

- start locator

Start a locator.

- start pulse

  Launch the Tanzu GemFire Pulse monitoring dashboard tool in the user's default system browser and navigates the user to the landing page (login page).

  > GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

- start server

  Start a Tanzu GemFire cache server process.

- start vsd

  Launch GemFire Visual Statistics Display (VSD) in a separate process.

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 [--groups=value(,value)*] [--members=value(,value)*]
```

Parameters, start gateway-receiver

Name	Description
--members	Name or ID of the members on which to start the Gateway Receiver.
--groups	Groups of members on which to start the Gateway Receiver.

Example Commands:

```
start gateway-receiver
start gateway-receiver --members=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 --members=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 about how to configure a gateway sender, see Configure Gateway Senders.

Note: By default, gateway senders are configured to start automatically. Manual restart introduces a risk of data loss; it is not intended for production systems.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
start gateway-sender --id=value [--groups=value(,value)*] [--members=value(,value)*]
[--clean-queues(=value)]
```

Parameters, start gateway-sender

Name	Description	Default Value
--id	*Required.* ID of the GatewaySender.	
--groups	Groups of members on which to start the Gateway Sender.	
--members	Members on which to start the Gateway Sender.	
--clean-queues	Option to clean existing queue at start of the Gateway Sender. This option is only applicable for Gateway Senders with enabled persistence.	false

Example Commands:

```
start gateway-sender --id=sender1-NY
start gateway-sender --id=sender1-NY --members=server1
start gateway-sender --id=sender1-NY --groups=MemberGroup1,MemberGroup2
start gateway-sender --id=sender1-NY --clean-queues
```

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 --members=pc13(30614)<v14>:44519
GatewaySender ln is started on member pc13(30614)<v14>:44519
gfsh>start gateway-sender --id=ln --groups=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 Tanzu GemFire MBeans through JConsole for an example of using JConsole with the Tanzu GemFire management and monitoring system.

Availability: Online or offline.

Syntax:

```
start jconsole [--interval=<seconds>] [--notile] [--version]
[--J<jconsole JVM options>]
```

Parameters, start jconsole

Name	Description	Default Value
--interval	Set the update interval to n seconds (default is 4 seconds). (Equivalent to JConsole's -interval=n)	4
--notile	Whether to initially tile windows for two or more connections. This parameter is passed as -notile to JConsole.	false
--plugin path	Directories or JAR files which are searched for JConsole plugins. The path should contain a provider-configuration file named:META-INF/services/com.sun.tools.jconsole.JConsolePlugin 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.	false
--J	Arguments passed to the JVM on which JConsole runs	

Example Commands:

```
gfsh>start jconsole --interval=8 --notile;
Running JDK JConsole

gfsh>start jconsole --version;
JConsole version "1.8.0_31-b01-1"
Java(TM) SE Runtime Environment (build 1.8.0_31-b01-1-11)
Java HotSpot(TM) 64-Bit Server VM (build 20.6-b01-11, 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 Tanzu 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)*]
```

Parameters, start jvisualvm

Name	Description
--J	VM-option passed to the spawned CacheServer VM. For example: --J=-Dfoo.bar=true for setting foo.bar to 'true'.

Example Commands:

```
start jvisualvm
```

Sample Output:

start locator

Start a locator.

- On Java 13 and earlier: When --max-heap is specified, gfsh activates the CMS garbage collector and sets -XX:CMSInitiatingOccupancyFraction to 60. If you do not want gfsh to set these GC properties, then use the -Xmx JVM option instead of using --max-heap.
- On Java 17 and later (except on Windows): JDK 17 defaults to the G1 garbage collector. However, because the Z garbage collector (ZGC) offers more predictable latencies and is easier to tune for GemFire heap LRU eviction, gfsh automatically activates ZGC on JDK 17. If you prefer to use the G1 garbage collector instead, use the

-XX:-UseZGC JVM option to deactivate ZGC and revert to G1. For information about which GC algorithm is best for your application, see Managing Heap Memory.

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.

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.

When --max-heap is specified, gfsh enables the CMS garbage collector and sets -XX:CMSInitiatingOccupancyFraction to 60. If you do not want gfsh to set these GC properties, then use the -Xmx JVM option instead. For more information, see Controlling Heap Use with the Resource Manager in *Managing Heap Memory*.

> The additional GC parameters introduced by the --max-heap option are not compatible with the G1 garbage collector.

Availability: Online or offline.

Syntax:

```
start locator --name=value [--bind-address=value] [--force(=value)]
 [--groups=value(,value)*] [--hostname-for-clients=value] [--classpath=value]
 [--locators=value] [--log-level=value] [--port=value] [--dir=value]
 [--disable-classloader-isolation(=value)] [--properties-file=value] [--security-properties-file=value]
 [--initial-heap=value] [--max-heap=value] [--connect(=value)] [--enable-cluster-configuration(=value)]
 [--load-cluster-configuration-from-dir(=value)] [--cluster-config-dir=value] [--redirect-output(=value)]
 [--http-service-port=value] [--http-service-bind-address=value]
 [--dry-run(=value)]
 [--J=value(,value)*]
```

Parameters, start locator

Name	Description	Default Value
--name	Name to be used for this Tanzu GemFire locator service. If not specified, gfsh generates a random name.	
--bind-address	IP address on which the locator will be bound.	bind to all addresses
--force	Whether to allow the PID file from a previous locator run to be overwritten.	false
--groups	Groups the locator will be a part of.	
--hostname-for-clients	Host name or IP address that will be sent to clients so they can connect to this locator.	uses bind-address
--classpath	Application classes to be added to the locator's class loader.	
--locators	List of locators used by this locator to join the appropriate Tanzu GemFire cluster.	
--log-level	Level of output logged to the locator log file. Possible values for log-level include: ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF.	
--port	Port the locator will listen on.	10334
--dir	Directory in which the Locator will be started and run.	./<locator-member-name>

Name	Description	Default Value
--disable-classloader-isolation	Causes the locator to be started without classloader isolation enabled.	false
--properties-file	Specify the gemfire.properties file for configuring the locator's cluster. The file's path should be absolute or relative to gfsh's working directory.	
--security-properties-file	The gfsecurity.properties file for configuring the Locator's security configuration in the cluster. The file's path can be absolute or relative to gfsh's working directory.	
--initial-heap	Size has the same format as the -Xmx/-Xms JVM options.	
--max-heap	Size has the same format as the -Xmx/-Xms JVM options. Note: The additional GC parameters introduced by the --max-heap option are not compatible with the G1 garbage collector.	
--connect	When connect is set to false, gfsh does not automatically connect to the locator which is started using this command.	true
--enable-cluster-configuration	Enables cluster configuration behavior where locators maintain configurations for all members of the cluster. See Overview of the Cluster Configuration Service.	true
--load-cluster-configuration-from-dir	Deprecated. Use gfsh import cluster-configuration for this functionality. Loads the cluster configuration from the shared-config directory. (When set to false, the configuration is loaded from the disk store of the internal, persistent region used by the locator to persist the configuration.)	false
--cluster-config-dir	Directory used by the cluster configuration service to store the cluster configuration on the filesystem	cluster-config
--redirect-output	When true, redirect standard output and standard error to the locator log file. If specified without a value, the value is set to true.	false
--http-service-port	Specifies the HTTP service port.	7070
--http-service-bind-address	Specifies the IP address to which the HTTP service will be bound.	the local host machine's address
--dry-run	When true, print the full Java command that would be used, given the startup options that were passed in. Does NOT start a locator. If specified without a value, the value is set to true.	false
--J	Argument passed to the JVM on which the Locator will run. For example, specifying --J=-Dfoo.bar=true sets property "foo.bar" to "true". Note: If the argument you are passing contains spaces or commas, enclose the option in single quotes. For example: start locator --name=locator1 --port=9009 --J='-Dgemfire.remote-locators=192.0.2.0[9009],192.0.2.1[9009]'	none

Example Commands:

```
start locator --name=locator1
```

start pulse

Launch the Tanzu GemFire Pulse monitoring dashboard tool in the user's default system browser and navigates the user to the landing page (login page).

> GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

For more information about Tanzu GemFire Pulse, see Tanzu GemFire Pulse.

Availability: Online or offline.

Syntax:

```
start pulse [--url=value]
```

Parameters, start pulse

Name	Description	Default Value
--url	URL of the Pulse Web application	http://localhost:7070/pulse

Example Commands:

```
start pulse
start pulse --url=http://gemfire.example.com:7070/pulse
```

Sample Output:

```
Launched Geode Pulse
```

start server

Start a Tanzu GemFire cache server process.

- On Java 13 and earlier: When --max-heap is specified, gfsh enables the CMS garbage collector and sets -XX:CMSInitiatingOccupancyFraction to 60. If you do not want gfsh to set these GC properties, then use the -Xmx JVM option instead of using --max-heap.
- On Java 17 and later (except on Windows):
  - JDK 17 defaults to the G1 garbage collector. However, because the Z garbage collector (ZGC) offers more predictable latencies and is easier to tune for GemFire heap LRU eviction, gfsh automatically activates ZGC on JDK 17. If you prefer to use the G1 garbage collector instead, use the -XX:-UseZGC JVM option to deactivate ZGC and revert to G1. For information about which GC algorithm is best for your application, see Managing Heap Memory.
  - If ZGC is activated and --eviction-heap-percentage is set, gfsh sets -XX:SoftMaxHeapSize to a byte value slightly lower than the eviction heap percentage. To specify a different soft max heap size, use the -XX:SoftMaxHeapSize=value JVM option. If you will use heap LRU eviction, we highly recommended that you set -XX:SoftMaxHeapSize, either directly or by setting the eviction heap percentage.

For more information, see Controlling Heap Use with the Resource Manager in *Managing Heap Memory*.

> The additional GC parameters introduced by the --max-heap option are not compatible with the G1 garbage collector.

Availability: Online or offline.

Syntax:

```
start server --name=value [--assign-buckets(=value)] [--bind-address=value]
    [--cache-xml-file=value] [--classpath=value] [--disable-default-server(=value)]
    [--disable-exit-when-out-of-memory(=value)] [--disable-classloader-isolation(=value)]
    [--enable-time-statistics(=value)] [--force(=value)]
    [--properties-file=value] [--security-properties-file=value] [--groups=value(,value)*]
    [--locators=value] [--locator-wait-time=value] [--log-level=value]
    [--memcached-port=value]
    [--memcached-protocol=value] [--rebalance(=value)] [--server-bind-address=value]
    [--server-port=value] [--spring-xml-location=value]
    [--statistic-archive-file=value] [--dir=value] [--initial-heap=value]
    [--max-heap=value] [--use-cluster-configuration(=value)] [--J=value(,value)*]
    [--critical-heap-percentage=value] [--critical-off-heap-percentage=value]
    [--eviction-heap-percentage=value] [--eviction-off-heap-percentage=value]
    [--hostname-for-clients=value] [--max-connections=value]
    [--message-time-to-live=value] [--max-message-count=value] [--max-threads=value]
    [--socket-buffer-size=value] [--lock-memory=value] [--off-heap-memory-size=value]
    [--start-rest-api=value] [--redirect-output(=value)]
    [--http-service-port=value] [--http-service-bind-address=value]
    [--username=value] [--password=value] [--dry-run(=value)]
```

Parameters, start server

Name	Description	Default Value
--name	Member name for this server. If not specified, gfsh generates a random name.	
--assign-buckets	Whether to assign buckets to the partitioned regions of the cache on server start.	false
--bind-address	The IP address on which the server will be bound.	binds to all local addresses
--cache-xml-file	Specifies the name of the XML file or resource to initialize the cache with when it is created.	
--classpath	Application classes to be added to the server's class loader after the core jar file.	
--disable-default-server	Whether the cache server will be started by default. If the parameter is specified without a value, the value is set to true. If set to true, the cache server acts as a peer.	false
--disable-exit-when-out-of-memory	Prevents the JVM from exiting when an OutOfMemoryError occurs.	false
--enable-time-statistics	Causes additional time-based statistics to be gathered for Tanzu GemFire operations.	true
--disable-classloader-isolation	Causes the server to be started without classloader isolation enabled. Deployed jars will be loaded without isolation.	false
--properties-file	The gemfire.properties file for configuring the server's cluster. The file's path can be absolute or relative to the gfsh working directory.	

Name	Description	Default Value
--security-properties-file	The gfsecurity.properties file for configuring the server's security configuration in the cluster. The file's path can be absolute or relative to gfsh directory.	
--groups	Groups the Cache Server will be a part of.	
--force	Whether to allow the PID file from a previous Cache Server run to be overwritten.	false
--locators	Sets the list of locators used by the Cache Server to join the appropriate Tanzu GemFire cluster.	
--locator-wait-time	Sets the number of seconds the server will wait for a locator to become available during startup before giving up.	0
--log-level	Sets the level of output logged to the Cache Server log file. Possible values for log-level include: ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF.	
--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	Overrides the bind-address on which this server will listen for client connections. Set this option in a multi-homed server environment to distinguish communications from clients. Setting a value of the empty string ("") uses the value of bind-address.	value of bind-address
--server-port	Port the Server will listen on for client connections.	40404
--spring-xml-location	Specifies the location of a Spring XML configuration file for bootstrapping and configuring a Tanzu GemFire Server. This configuration file can exist on the class path (default) or any location supported by Spring's Resource(Loader) location specifiers (for example, classpath:, file:, etc). ResourceLoader is described in the Spring documentation.	
--rebalance	Whether to initiate rebalancing across the Tanzu GemFire cluster.	false
--dir	Specify the directory in which the server will run. This directory is written to the location where you started gfsh.	If not specified, the directory is named after the server.
--statistic-archive-file	The file that statistic samples are written to. For example: "StatisticsArchiveFile.gfs".	{workingDir}/stats/{name}.gfs
--initial-heap	Initial size of the heap in the same format as the JVM -Xms parameter.	
--max-heap	Maximum size of the heap in the same format as the JVM -Xmx parameter. Note: The additional GC parameters introduced by the --max-heap option are not compatible with the G1 garbage collector.	
--J	Argument passed to the JVM on which the Cache Server will run. For example, --J=-Dfoo.bar=true will set the property "foo.bar" to "true". If the argument you are passing contains spaces or commas, enclose the option in single quotes.	

Name	Description	Default Value
--use-cluster-configuration	Specifies whether the server requests a cluster configuration from the locator. See Overview of the Cluster Configuration Service.	true
--critical-heap-percentage	Set the 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. Past the threshold, operations that require heap space will throw a LowMemoryException. This feature requires additional VM flags to perform properly; you must set --initial-heap and --max-heap or the corresponding JVM properties to use this threshold. You must also set --max-heap and --initial-heap to the same value.	0 (no critical heap threshold enforced)
--critical-off-heap-percentage	The percentage of off-heap memory used at or above which the cache is considered in danger of becoming inoperable due to out of memory exceptions. Past the threshold, operations that require heap space will throw a LowMemoryException.	0 (no critical off-heap threshold enforced)
--eviction-heap-percentage	Set the percentage of heap at or above which the eviction should begin on Regions configured for HeapLRU eviction. Changing this value may cause eviction to begin immediately. 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; you must set --initial-heap and --max-heap or the corresponding JVM properties to use this threshold. You must also set --max-heap and --initial-heap to the same value.	• 0, if no region is configured with heap eviction • If critical-heap-percentage is set to a non-zero value, 5% less than that value. • 80%, if critical-heap-percentage is not configured.
--eviction-off-heap-percentage	The percentage of off-heap memory used at or above which the eviction should begin on regions configured for off-heap and HeapLRU eviction. Changing this value may cause eviction to begin immediately. Only one change to this attribute or critical off-heap percentage will be allowed at any given time, and its effect will be fully realized before the next change is allowed.	• 0, if no region is configured with heap eviction • If critical-off-heap-percentage is set to a non-zero value, 5% less than that value. • 80%, if critical-off-heap-percentage is not configured.

Name	Description	Default Value
--hostname-for-clients	Sets the IP address or host name that a locator will provide to clients. Clients use the address to connect to a server. Set this value when clients use a different address to connect with the server than the bind-address, as those clients might with servers in a private cloud or multi-homed environment. Not specifying this option or setting this option to the empty string ("") causes the bind-address to be given to clients.	
--max-connections	Sets the maximum number of client connections allowed. When the maximum is reached the cache server will stop accepting connections.	
--message-time-to-live	Sets the time (in seconds) after which a message in the client queue will expire.	
--max-message-count	Sets maximum number of messages that can be enqueued in a client-queue.	
--max-threads	Sets the maximum number of threads allowed in this cache server to service client requests. The default of 0 causes the cache server to dedicate a thread for every client connection. When client-server TLS/SSL is configured, values other than the default are not supported.	
--socket-buffer-size	Sets the buffer size in bytes of the socket connection for this CacheServer. The default is 32768 bytes.	
--lock-memory	(Linux only) When true, the member's heap and off-heap memory are locked in RAM, preventing them from being paged to disk. You must increase the related ulimit operating system resource to allow the OS to lock memory chunks of sufficient size.	false
--off-heap-memory-size	The integer quantity of off-heap memory to be used for storing region values. Specified in Gigabytes with a "g" suffix, or Megabytes with an "m" suffix. For example, allocate a 2 Gigabyte off-heap space with --off-heap-memory-size=2g. The default value of 0 does not use any off-heap memory.	0
--start-rest-api	When true, starts the REST API service.	false
--redirect-output	When true, redirect standard output and standard error to the server log file. If specified without a value, the value is set to true.	false
--http-service-port	Specifies the HTTP service port.	7070
--http-service-bind-address	Specifies the IP address to which the HTTP service will be bound.	the local host machine's address
--username --user	The user name of the credential to use in authenticating to the cluster. When specified, if the --password option is not also specified, then gfsh will prompt for the password.	
--password	The password portion of the credential to use in authenticating to the cluster.	
--dry-run	When true, print the full Java command that would be used, given the startup options that were passed in. Does NOT start a server. If specified without a value, the value is set to true.	false

Examples

```
gfsh>start server --name=server1
gfsh>start server --name=server2 --server-port=40405
```

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 [--file=value(nullvalue)*]
```

Parameters, start vsd

Name	Description
--file	File or directory from which to read the statistics archives.

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

This topic describes the status command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to check the status of the cluster configuration service and VMware Tanzu GemFire member processes, including locators, gateway receivers, gateway senders, and servers.

- status cluster-config-service

  Displays the status of the cluster configuration service.

- status gateway-receiver

  Display the status of the specified gateway receiver.

- status gateway-sender

  Display the status of the specified gateway sender.

- status locator

  Displays the status of the specified locator.

- status redundancy

  Displays the redundancy status of partitioned regions.

- status server

  Display the status of the specified Tanzu GemFire cache server.

status cluster-config-service

Displays the status of the cluster configuration service.

Displays the status of cluster configuration service on all the locators where enable-cluster-configuration is set to true.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
status cluster-config-service
```

Example Commands:

```
status cluster-config-service
```

Sample Output:

```
gfsh>status cluster-config-service
Status of shared configuration on locators

  Name   | Status
-------- | -------
locator8 | RUNNING
```

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 [--groups=value(,value)*] [--members=value(,value)*]
```

Name	Description
--groups	Groups of Gateway Receivers for which to display status.
--members	Name or ID of the Gateway Receivers for which to display status.

Table 1. Status Gateway-Receiver Parameters

Example Commands:

```
status gateway-receiver --groups=LN-Group1
status gateway-receiver --members=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 --members=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 [--groups=value(,value)*]
[--members=value(,value)*]
```

Name	Description
--id	*Required.* ID of the Gateway Sender.
--groups	Groups of Gateway Senders for which to display status. Comma separated list for multiple member groups.
--members	Name/ID of the Gateway Senders for which to display status.

Table 2. Status Gateway-Sender Parameters

Example Commands:

```
status gateway-receiver receiver1-LN --groups=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 --members=pc13(8151)<v2>:7411
Member                |  Type  | Runtime Policy | Status
-------------------  | ------ | -------------- | -----------
 pc13(8151)<v2>:7411 | Serial | Secondary      | Not Running

gfsh>status gateway-sender --id=ln_Serial --members=pc13(8151)<v2>:7411
Member                |  Type  | Runtime Policy | Status
------------------- -| ------ | -------------- | -------
 pc13(8151)<v2>:7411 | Serial | Primary        | Running

gfsh>status gateway-sender --id=ln_Serial --groups=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. If you want to obtain the status of a locator while you are offline, use the --dir option.

Syntax:

```
status locator [--name=value] [--host=value] [--port=value] [--dir=value] [--security-properties-file=value]
```

Name	Description	Default Value
--name	Name/ID of the locator for which to display status. You must be connected to the JMX Manager to use this option. Can be used to obtain status of remote locators. See Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS.	
--host	Hostname or IP address on which the Locator is running.	
--port	Port on which the locator is listening.	10334
--dir	Directory in which the locator was started.	current directory
--security-properties-file	The properties file for configuring SSL to connect to the SSL-enabled Locator. The file's path can be absolute or relative to the gfsh directory.	current directory

Table 3. Status Locator Parameters

Example Commands:

```
status locator
status locator --name=locator1
```

status redundancy

Display the redundancy status of partitioned regions.

The default is to display the status of all partitioned regions.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
status redundancy [--include-region=value(,value)*] [--exclude-region=value(,value)*]
```

Name	Description
--include-region	Partitioned Region paths to be included in redundancy status output. Includes take precedence over excludes.
--exclude-region	Partitioned Region paths to be excluded in redundancy status output.

Example Commands:

```
status redundancy
status redundancy --include-region=/region3,/region2 --exclude-region=/region1
```

Sample Output:

```
status redundancy --include-region=/region3,/region2 --exclude-region=/region1

Number of regions with zero redundant copies = 0
Number of regions with partially satisfied redundancy = 1
Number of regions with fully satisfied redundancy = 1

Redundancy is partially satisfied for regions:
  region3 redundancy status: NOT_SATISFIED. Desired redundancy is 2 and actual redundancy is 1.

Redundancy is fully satisfied for regions:
  region2 redundancy status: SATISFIED. Desired redundancy is 1 and actual redundancy is 1.
```

status server

Display the status of the specified Tanzu GemFire cache server.

Availability: Online or offline. If you want to obtain the status of a server while you are offline, use the --dir option.

Syntax:

```
status server [--name=value] [--dir=value]
```

Name	Description	Default Value
--name	Name or ID of the Cache Server for which to display status. You must be connected to the JMX Manager to use this option. Can be used to obtain status of remote servers. See Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS.	
--dir	Directory in which the Tanzu GemFire Cache Server was started.	current directory

Table 4. Status Server Parameters

Example Commands:

```
status server
status server --name=server1
```

stop

This topic describes the stop command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to stop gateway receivers, gateway senders, locators, and servers.

- stop gateway-receiver

  Stop the gateway receiver on a member or members.

- stop gateway-sender

  Stop a gateway sender with a given id on a specified member or members of a specified member group.

- stop locator

  Stop a locator.

- stop server

  Stop a Tanzu GemFire cache server.

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 [--groups=value(,value)*] [--members=value(,value)*]
```

Name	Description
--groups	Groups of members on which to stop the Gateway Receiver. Use a comma-separated list for multiple member groups.
--members	Name or ID of the members on which to stop the Gateway Receiver.

Table 1. Stop Gateway-Receiver Parameters

Example Commands:

```
stop gateway-receiver --members=receiver1-LN
stop gateway-receiver --groups=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 --members=pc13(2266)<v14>:36579
GatewayReceiver is stopped on member pc13(2266)<v14>:36579

gfsh>stop gateway-receiver --groups=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.

CAUTION: Use caution with the stop gateway-sender command (or equivalent GatewaySender.stop() API) on parallel gateway senders. Instead of stopping an individual parallel gateway sender on a member, we recommend shutting down the entire member to ensure that proper failover of partition region events to other gateway sender members. Using this command on an individual parallel gateway sender can occur in event loss. See Stopping Gateway Senders for more details.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
stop gateway-sender --id=value [--groups=value(,value)*] [--members=value(,value)*]
```

Name	Description
--id	*Required.* ID of the Gateway Sender.
--groups	Groups of members on which to stop the Gateway Sender.
--members	Name/Id of the members on which to stop the Gateway Sender.

Table 2. Stop Gateway-Sender Parameters

Example Commands:

```
stop gateway-sender --id=ln --members=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 --members=pc13(5174)<v14>:17819
GatewaySender ln is stopped on member pc13(5174)<v14>:17819

gfsh>stop gateway-sender --id=ln --groups=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.

> One of the command line options --name or --dir must be specified to identify the locator to be stopped.

Availability: Online or offline. If you want to stop a locator while you are offline, use the --dir option.

Syntax:

```
stop locator --name=value | --dir=value
```

Name	Description	Default Value
--name	The Tanzu GemFire member name of the locator to stop. You must be connected to the JMX Manager to use this option. Can be used to stop remote locators. See Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS.	
--dir	Directory in which the locator was started.	current directory

Table 3. Stop Locator Parameters

Example Commands:

```
stop locator --name=locator3
```

Sample Output:

```
gfsh>stop locator --name=locator3
Stopping Locator running in /Users/test/locator3 on 192.0.2.0[10334] as locator3...
Process ID: 71531
Log File: /Users/test/locator3/locator3.log
...
No longer connected to 192.0.2.0[1099].
```

```
gfsh>stop locator --dir=loc2
Stopping Locator running in /Users/test/loc2 on 192.0.2.0[10334] as loc2...
Process ID: 71714
Log File: /Users/test/loc2/loc2.log
...
No longer connected to 192.0.2.0[1099].
```

stop server

Stop a Tanzu GemFire cache server.

Availability: Online or offline. If you want to stop a cache server while you are offline, use the --dir option.

Syntax:

```
stop server [--name=value] [--dir=value]
```

Name	Description	Default Value
--name	Name/Id of the Tanzu GemFire Cache Server to stop. You must be connected to the JMX Manager to use this option. Can be used to stop remote servers. See Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS.	
--dir	Directory in which the Tanzu GemFire Cache Server was started.	current directory

Table 4. Stop Server Parameters

Example Commands:

```
stop server --name=server1
stop server --dir=server1
```

undeploy

This topic describes the undeploy command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to undeploy the JAR files that were deployed on members or groups using deploy command.

If --jars is not specified, the command will undeploy all deployed JARs. If --groups is not specified, the command applies to the entire cluster. Note that this command cannot unload the classes that were loaded during deployment. You must restart the members to unload the classes that were loaded during deployment.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
undeploy [--jars=value(,value)*] [--groups=value(,value)*]
```

Name	Description	Default Value
--groups	Groups from which the specified JARs will be undeployed.	Undeploy will occur on all members
--jars	JAR or JARs to be undeployed.	All JARs will be undeployed

Table 1. Undeploy Parameters

Example Commands:

```
undeploy --jars=domain-objects.jar
undeploy --groups=Group1
```

Sample Output:

```
gfsh>undeploy --jars=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 --groups=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

This topic describes the upgrade offline-disk-store command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to upgrade offline disk stores from the format used in Tanzu GemFire releases prior to 10.1 to the segmented disk store type introduced in Tanzu GemFire version 10.1. See Defining and Configuring Disk Stores for more information about the benefits of segmented disk stores.

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 --source-disk-dirs=value(,value)*
[--destination-disk-dirs=value(,value)] [--segments=value]
[--max-oplog-size=value] [--J=value(,value)*]
```

Name	Description	Default Value
--name	*Required.* Name of the offline disk store to be upgraded.	

Name	Description	Default Value
--source-disk-dirs	*Required.* Directories where data for the disk store was previously written.	
--destination-disk-dirs	Directories where data for the disk store is to be written.	source-disk-dirs
--segments	Number of segments to upgrade to.	Current number of available CPUs on the this host.
--delete-source	Delete source from disk store on successful upgrade.	false
--max-oplog-size	Maximum size, in megabytes, of the oplogs created by the upgrade. A value of -1 indicates that there is no maximum size limit for oplogs.	-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 --source-disk-dirs=/Disks/DiskStore1 --segments=32 --delete-source=true
upgrade offline-disk-store --name=DiskStore1 --source-disk-dirs=/Disks/DiskStore1 --segments=32 --delete-source=true --J=-DX
mx512
```

validate offline-disk-store

This topic describes the validate offline-disk-store command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to validate offline disk stores.

Availability: Offline.

Syntax:

```
validate offline-disk-store --name=value --disk-dirs=value(,value)*
```

Name	Description
--name	*Required.* Name of the disk store to be validated.
--disk-dirs	*Required.* Directories where data for the disk store was previously written.

Table 1. Validate Offline-Disk-Store Parameters

Example Commands:

```
validate offline-disk-store --name=DiskStore2 --disk-dirs=data/dir3,data/dir4
```

version

This topic describes the version command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to display product version information.

Availability: Online or offline.

Syntax:

```
version [--full]
```

Name	Description	Defalut Value
--full	Show the full version information.	false

Table 1. Version Parameters

Example Commands:

```
version
version --full
```

Sample Output:

```
gfsh>version
v8.0.0


gfsh>version --full
Java version:   8.0.0 build 48319 07/31/2014 17:26:09 PDT javac 1.8.0_1
Native version: native code unavailable
Source revision: 48319
Source repository: gemfire/branches/cedar_dev_Oct12
Running on: /192.0.2.0, 1 cpu(s), amd64 Linux 2.6.32-38-generic
```

wan-copy region

This topic describes the wan-copy region command in gfsh, the VMware Tanzu GemFire command-line interface.

Use this command to copy the entries of one or more regions in a WAN site onto the same region in another WAN site, using a gateway sender.

This command copies the entries for one or more regions from a WAN site to another by putting them in batches of configurable size that are sent to the remote site by the selected gateway sender. Batch size is specified as number of entries per batch.

Callbacks (cache listeners, replication to other WAN sites) are not executed in the remote WAN site for the entries copied.

The main uses of this command are:

- Recovery of a WAN site after a disaster where the failed WAN site must be put into service again using the data from another WAN site.
- Adding a new WAN site to a Tanzu GemFire system where the data in the new WAN site must initially be loaded from an existing WAN site.

Requirements:

The wan-copy region command requires the following:

- A gateway sender is configured and running on the source WAN site.
- A gateway receiver is configured and running on the remote WAN site.
- The region the data will be copied to is created on the remote WAN site.
- You must be connected in gfsh to a JMX Manager member.

Syntax:

```
wan-copy region --region=value [--sender-id=value] [--max-rate=value] [--batch-size=value] [--cancel][--simulate][--background][--wan-copy-id]
```

Parameters

Name	Description
region	*Required*. The region or regions where the data will be copied. You can define multiple regions. There are three methods for specifying the region: wildcard *, wildcard ?, comma separated list of regions. See Region Expression Examples table.
sender-id	*Optional*. The gateway sender to use to copy the region entries. If left blank, the wan-copy region command copies region entries to all sender-ids associated with the region. As this parameter is optional, you can identify the regions you want to copy to other cluster(s) without listing all sender-ids
max-rate	The maximum copy rate in entries per second. Specify a maximum copy rate to avoid overloading the sending or receiving WAN sites. If the sender is parallel, the maximum rate limit is applied to each server hosting bucket for the region to be copied. The default is 0. (unlimited)
batch-size	The size of the batches, in number of entries, to be used to copy the region entries. The default value is 1000.
cancel	Cancel all current copies in progress. Cancel a running wan-copy region command for the specified sender and region. If the sender-id and region passed are both "*", then all running wan-copy region commands are canceled. To cancel a specific wan-copy region command, use the –cancel flag along with the –wan-copy-id flag. This cancels the copy executions associated with that wan-copy-id even if the copy executions are not started.
background	Allows the wan-copy command to run in the background, to accommodate large groups of regions that might take considerable time to copy. A wan-copy-id is returned that can be used with other commands to manage the wan-copy region process while it continues running in the background.
simulate	Allows you to test an expression to ensure you are including the correct regions and sender IDs before executing the actual command. Returns a list of region/sender ID pairs that will be copied by the wan-copy region command.
wan-copy-id	To identify and manage wan-copy region commands while they are in progress, each execution is associated with a wan-copy-id. This ID can be used in subsequent commands to check the status of or cancel a wan-copy region command. The wan-copy-id is returned with –background flag and can also be retrieved using the list wan-copy region command.

> The --simulate flag cannot be used with the --cancel or --background flags. Attempting to set either of these flags results in an error.

Example Commands

```
wan-copy region --region=myRegion--sender-id=mySender --max-rate=1000 --batch-size=100
```

```
wan-copy region --region=/overload --sender-id=sender1 --cancel
```

```
wan-copy region --region=* --sender-id=* --cancel
```

```
wan-copy region --region=myRegion --cancel --wan-copy-id=myID
```

```
wan-copy region --region=region1*
```

```
wan-copy region --region=region1?  --sender-id=ny*
```

```
wan-copy region --region=region1,region2 --sender-id=ny1,ny2*
```

Example Output 1

This example shows the output for a wan-copy command which runs on 3 regions which all copy successfully with a duration of 6 seconds.

```
wan-copy region --region=testSuccessfulExecution_PR* --sender-id=B* --batch-size=20

3 of 3 Regions copied.
0 hrs 0 min 6 sec

Region Name: testSuccessfulExecution_PR3
Sender ID: B

       Member          | Status | Message
----------------------- | ------ | ------------------
dyn-13(79844)<v1>:53464 | OK     | Entries copied: 33
dyn-13(79845)<v2>:41794 | OK     | Entries copied: 33
dyn-13(79846)<v3>:52035 | OK     | Entries copied: 34

Region Name: testSuccessfulExecution_PR1
Sender ID: B

       Member          | Status | Message
----------------------- | ------ | ------------------
dyn-13(79846)<v3>:52035 | OK     | Entries copied: 33
dyn-13(79844)<v1>:53464 | OK     | Entries copied: 33
dyn-13(79845)<v2>:41794 | OK     | Entries copied: 34

Region Name: testSuccessfulExecution_PR2
Sender ID: B

       Member          | Status | Message
----------------------- | ------ | ------------------
dyn-13(79844)<v1>:53464 | OK     | Entries copied: 34
dyn-13(79845)<v2>:41794 | OK     | Entries copied: 33
dyn-13(79846)<v3>:52035 | OK     | Entries copied: 33
```

Example Output 2

This example executes wan-copy on 4 regions, however when this wan-copy command is on-going, another terminal executes the cancel command causing the operation to cancel before all entries are copied.

```
wan-copy region --region=region1* --sender-id=ny*

2 of 4 regions copied.
Elapsed Time: 2 hrs 45 min 33 sec

Region Name: region1-east
Sender Id: ny-east
Status: FINISHED

Member      | Status | Message
----------- | ------ | -------------------
server-ln-2 | OK     | Entries copied: 252
server-ln-1 | OK     | Entries copied: 248

Region Name: region1-west
Sender Id: ny-west
Status: FINISHED

Member      | Status | Message
----------- | ------ | -------------------
server-ln-2 | OK     | Entries copied: 85
server-ln-1 | OK     | Entries copied: 85


Region Name: region1-north
Sender Id: ny-north
Status: CANCELED

Member      | Status   | Message
----------- | -------- | ---------------------------------------------------
server-ln-2 | CANCELED | Operation canceled before having copied all entries
server-ln-1 | CANCELED | Operation canceled before having copied all entries


Region Name: region1-south
Sender Id: ny-south
Status: CANCELED

Member      | Status   | Message
----------- | -------- | ---------------------------------------------------
server-ln-2 | CANCELED | Operation canceled before having copied all entries
server-ln-1 | CANCELED | Execution canceled before having copied all entries
```

Example 3

This example executes wan-copy with the --cancel flag. This impacts the output

```
wan-copy region --region=region1* --sender-id-ny* --cancel

Wan-copy-id=cc09e6a1-99a7-4515-ba7c-3ec2caa15446 is canceled
Attempted to copy 4 region(s)

Region Name: region1-east
Sender Id: ny-east

Member      | Status  | Message
------------|---------|-------------------------------------------------------------------------------
server-ln-1 | OK      | No running command to be canceled for region region1-east and sender ny-east
server-ln-2 | OK      | No running command to be canceled for region region1-east and sender ny-east

Region Name: region1-west
Sender Id: ny-west

Member      | Status  | Message
------------|---------|-------------------------------------------------------------------------------
server-ln-1 | OK      | No running command to be canceled for region region1-west and sender ny-west
server-ln-2 | OK      | No running command to be canceled for region region1-west and sender ny-west

Region Name: region1-north
Sender Id: ny-north

Member      | Status  | Message
------------|---------|--------------------
server-ln-1 | OK      | Execution canceled
server-ln-2 | OK      | Execution canceled

Region Name: region1-south
Sender Id: ny-south

Member      | Status  | Message
------------|---------|--------------------
server-ln-1 | OK      | Execution canceled
server-ln-2 | OK      | Execution canceled
```

Example 4

```
wan-copy region --region=* --cancel --wan-copy-id=cc09e6a1-99a7-4515-ba7c-3ec2caa15446


Wan-copy-id=cc09e6a1-99a7-4515-ba7c-3ec2caa15446 is canceled
Attempted to copy 4 region(s)

Region Name: region1-east
Sender Id: ny-east

Member      | Status  | Message
------------|---------|-------------------------------------------------------------------------------
server-ln-1 | OK      | No running command to be canceled for region region1-east and sender ny-east
server-ln-2 | OK      | No running command to be canceled for region region1-east and sender ny-east

Region Name: region1-west
Sender Id: ny-west

Member      | Status  | Message
------------|---------|-------------------------------------------------------------------------------
server-ln-1 | OK      | No running command to be canceled for region region1-west and sender ny-west
server-ln-2 | OK      | No running command to be canceled for region region1-west and sender ny-west

Region Name: region1-north
Sender Id: ny-north

Member      | Status  | Message
------------|---------|--------------------
server-ln-1 | OK      | Execution canceled
server-ln-2 | OK      | Execution canceled

Region Name: region1-south
Sender Id: ny-south

Member      | Status  | Message
------------|---------|--------------------
server-ln-1 | OK      | Execution canceled
server-ln-2 | OK      | Execution canceled
```

Error Messages

Example of wan-copy region with an invalid region:

```
gfsh> wan-copy region --region=/regionX --sender-id=sender1
      Member     | Status | Message
   -------------- | ------ | -------------------------
   server-sender  | ERROR  | Region /regionX not found
   server-sender2 | ERROR  | Region /regionX not found
   server-sender3 | ERROR  | Region /regionX not found
```

Example of wan-copy region with a stopped gateway sender:

```
gfsh> wan-copy region --region=/region1 --sender-id=sender1
      Member     | Status | Message
   -------------- | ------ | ----------------------------
   server-sender  | ERROR  | Sender sender1 is not running
   server-sender2 | ERROR  | Sender sender1 is not running
   server-sender3 | ERROR  | Sender sender1 is not running
```

Example of cancel of wan-copy region when no command is running:

```
gfsh> wan-copy region --region=/region1 --sender-id=sender1 --cancel
      Member     | Status | Message
   -------------- | ------ | ----------------------------------------------------------------------------------------
   server-sender2 | ERROR  | No running command to be canceled for region /region1 and sender sender1
   server-sender  | ERROR  | No running command to be canceled for region /region1 and sender sender1
   server-sender3 | ERROR  | No running command to be canceled for region /region1 and sender sender1
```

Example of cancel of all running wan-copy region commands:

```
gfsh> wan-copy region --region=* --sender-id=* --cancel
      Member     | Status | Message
   -------------- | ------ | ----------------------------------------------------------------------------------------
   server-sender2 | OK     | Executions canceled: [(myRegion,mySender1), (myRegion,mySender)]
   server-sender  | OK     | Executions canceled: [(myRegion,mySender1), (myRegion,mySender)]
   server-sender3 | OK     | Executions canceled: [(myRegion,mySender1), (myRegion,mySender)]
```

Example of when the –cancel flag is not set in conjunction with the –wan-copy-id flag

```
wan-copy region --region=* --wan-copy-id=cc09e6a1-99a7-4515-ba7c-3ec2caa15446

Error: wan-copy-id parameter is for cancel or describe command.
```

Region Expression Examples

Regions	Expression	Matching Regions
regionA regionB regionC planes trains automobiles	wan-copy region –region=*	regionA regionB regionC planes trains automobiles
regionA regionB regionC planes trains automobiles	wan-copy region –region=region*	regionA regionB regionC
cats dogs horses bats ats	wan-copy region –region=?ats	cats bats
regionA regionB regionC planes trains automobiles	wan-copy region –region=regionA, planes, automobiles	regionA planes automobiles
regionA regionB regionC planes trains automobiles	wan-copy region –region=region*, planes	regionA regionB regionC planes

Regions	Expression	Matching Regions
regionA regionB regionC planes trains automobiles cats dogs horses bats	wan-copy region –region=region*, ?ats	regionA regionB regionC cats bats
regionA regionB regionC planesregion trains automobiles	wan-copy region –region=*region*	regionA regionB regionC planesregion
regionA regionB regionC a1b12c abc trains automobiles	wan-copy region –region=a?b*c	a1b12c
region` region[] region[1]	wan-copy region –region=region*] or wan-copy region –region=region[* or wan-copy region –region=region[*]	region[] region[1]
region` region[] region[1]	wan-copy region –region=region?] or wan-copy region –region=region[? or wan-copy region –region=region[?] or wan-copy region –region=region[] or wan-copy region –region=region[1]	Found nothing. This is a limitation.

Creating and Running gfsh Command Scripts

This topic explains how to create and run command scripts using gfsh, the Tanzu GemFire command-line.

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 command. For example:

```
gfsh run --file=mycommands.gfsh
```

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.

You may find your command history file to 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 any user running gfsh. You can also export a history file with the history --file=your_file_name command.

When a user runs start server or start locator from gfsh without specifying the member name, gfsh automatically uses a randomly selected member name. Use this for automation.

Running gfsh Commands on the OS Command Line

This topic explains that you can run some VMware Tanzu GemFire command-line interface (gfsh) commands directly from your operating system's prompt by preceding the command with gfsh. Use this for Unix shell or Windows batch scripting. For example:

```
$ gfsh start locator --name=locator2 --port=10335
```

To see which gfsh commands are available directly on the prompt:

```
$ gfsh help
```

Running Multiple gfsh Commands on the OS Command Line

To run multiple commands directly on the command line, use the -e option followed by the gfsh command within quote marks. For example:

```
prompt>gfsh -e "start locator --name=locator1" -e "start server --name=server1"

prompt>gfsh -e "start jconsole"

prompt>gfsh -e "connect --locator=remotehost[10334]" -e "rebalance" -e "gc"
```

Specifying JSON within Command-Line Options

This topic explains how to specify JSON within command-line options when using gfsh, the VMware Tanzu GemFire command-line interface.

Some gfsh commands allow JSON specifications within some options. These JSON strings specify initialization properties in the form of key/value pairs.

A JSON string is appended to a fully qualified class name, for those classes that implement the Declarable interface. The parameter to the interface's init method is an object that contains initialization properties. The JSON string is parsed and broken into a set of property names and values that define the object.

The specification of a cache loader class in the gfsh create region command may have a JSON specification. For example,

```
gfsh>create region --name=region1 --type=PARTITION \
  --cache-loader=com.example.Setup{'k1':'v1','k2':'v2','k3':'v3'}
```

The JSON string in this example is

```
{'k1':'v1','k2':'v2','k3':'v3'}
```

The JSON string is surrounded by curly braces, and the pairs are separated by commas.
Each pair is separated by a colon. Each of the two strings that define a pair is enclosed with single or

double quote characters. Be consistent throughout the JSON string in the use of either single or double quote characters.

Space characters separate gfsh command-line options, and the tokenizing of the options occurs before the parsing of JSON strings. If the value within the command-line option contains a space, delimit it with double quote marks.

As an example, assume that there is a single pair, and the value of the pair is value with space. The right-hand side of the --cache-loader example option becomes

```
gfsh>create region --name=region2 --type=PARTITION \
  --cache-loader="com.example.Setup{'k1':'value with space'}"
```

To incorporate a single quote character within a value, escape it with two backslash characters. For example,

```
gfsh>create region --name=region3 --type=PARTITION \
  --cache-loader="com.example.Setup{'k1':'property\\'s value'}"
```

The value of the pair is property's value.

Mapping cache.xml Elements to gfsh Configuration Commands

This topic maps cache.xml elements to VMware Tanzu GemFire command-line interface (gfsh) commands.

You can configure a Tanzu GemFire cluster using either cache.xml files, or you can use gfsh and the cluster configuration service to configure a cluster. This table maps cache.xml elements to the gfsh commands that configure and manage a cluster.

cache.xml Element	gfsh Command
<cache>, <cache-server>	• start server • status server • stop server • alter runtime
<async-event-queue>	• alter async-event-queue • create async-event-queue • destroy async-event-queue • list async-event-queues • resume async-event-queue-dispatching
<pdx>	• configure pdx
<region>	• create region • alter region • destroy region • describe region • list regions • rebalance

cache.xml Element	gfsh Command
<index>	• create index • destroy index • list indexes
<disk-store>	• create disk-store • alter disk-store • backup disk-store • compact disk-store • compact offline-disk-store • describe disk-store • describe offline-disk-store • destroy disk-store • list disk-stores • revoke missing-disk-store • show missing-disk-stores • upgrade offline-disk-store • validate offline-disk-store
<query-config-service>	• alter query-service

Gemcached

Gemcached is deprecated and will be removed in upcoming releases.

This topic provides links to documentation about Gemcached in VMware Tanzu GemFire.

Gemcached is a Tanzu GemFire adapter that allows Memcached clients to communicate with a Tanzu 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.

- 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.

- 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 Tanzu GemFire addresses these challenges.

How Gemcached Works

Gemcached is deprecated and will be removed in upcoming releases.

This topic explains how VMware Tanzu GemFire 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 VMware Tanzu GemFire, you embed a Gemcached server within a Tanzu 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. Tanzu GemFire manages the distribution and access to data among the embedded Gemcached servers.

As shown in Gemcached Architecture, 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: Gemcached Architecture

Memcached clients use the memcached API to read and write data that is stored in memcached servers; therefore, client-side Tanzu GemFire features are not available to these clients. Gemcached servers, however, can use Tanzu GemFire's server-side features and API. These features include the following. (For more detail, see Advantages of Gemcached over Memcached.)

- Data consistency and scalability.
- High availability.
- Read-through, write through, and write behind to a database, implemented from within the distributed Tanzu 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.

- Configurable expiration of cached data.
- Configurable eviction of cached data.

Deploying and Configuring a Gemcached Server

Gemcached is deprecated and will be removed in upcoming releases.

This topic explains how to configure and deploy an embedded VMware Tanzu GemFire Gemcached server. You can configure and deploy Gemcached servers in a Java class or by using the gfsh command-line interface.

Embedding a Gemcached server in a Tanzu GemFire Java Application

The org.apache.geode.memcached package contains a single class, GemFireMemcachedServer (see the Tanzu GemFire Javadocs.) Use this class to configure and embed a Gemcached server in a Tanzu 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();
```

> 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.

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
```

> 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 1. 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 memcached-port parameter is omitted, the Gemcached server does not start.
memcached-protocol	Memcached supports both ASCII and binary communication protocols. (See Memcached protocol By default, Gemcached uses the ASCII protocol. Set one of the following values: • ASCII (default) • BINARY

Table 1. Gemcached Properties

Advantages of Gemcached over Memcached

Gemcached is deprecated and will be removed in upcoming releases.

This topic explains how the advantages of VMware Tanzu GemFire Gemcached over memcached.

The standard memcached architecture has inherent architectural challenges that make memcached applications difficult to write, maintain, and scale. Using Gemcached with Tanzu 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 Tanzu 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 Tanzu 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 Tanzu 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. Tanzu 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 Tanzu 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 Tanzu GemFire's standard features.

Pulse

VMware Tanzu GemFire Pulse is a Web Application that provides a graphical dashboard for monitoring vital, real-time health and performance of Tanzu GemFire clusters, members, and regions.

> Tanzu GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

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 Tanzu GemFire JMX manager to provide a complete view of your Tanzu 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, Tanzu GemFire Pulse runs in an embedded container within a Tanzu 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 Tanzu GemFire clusters. Hosting Pulse on an application server also enables you to use SSL for accessing the application.

- Pulse System Requirements

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

- Running Pulse in Embedded Mode (Quick Start)

  Use Pulse in embedded mode to monitor a Tanzu GemFire deployment directly from a Tanzu 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 Tanzu GemFire system of your choice.

- 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.

- 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 Tanzu GemFire JMX Manager node, the Pulse Web application itself may also need to authenticate itself to the Tanzu GemFire JMX Manager node on startup.

- Using Pulse Views

  Pulse provides a variety of different views to help you monitor Tanzu GemFire clusters, members, and regions.

Running Pulse in Embedded Mode (Quick Start)

This topic provides an overview to running VMware Tanzu GemFire Pulse in embedded mode. Use Pulse in embedded mode to monitor a Tanzu GemFire deployment directly from a Tanzu 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 Tanzu GemFire system of your choice.

> Tanzu GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

To run Pulse in embedded mode:

1. Configure a Tanzu GemFire member to run as a JMX Manager node, specifying the HTTP port on which you will access the Pulse Web application (port 7070 by default). For example, the following command starts a Tanzu GemFire locator as a JMX Manager node, using the default HTTP port 7070 for the Pulse application:

   ```
   gfsh
   gfsh> start locator --name=loc1
   ```

   > Tanzu 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 the HTTP service that hosts the Pulse application, include the --J=-Dgemfire.http-service-port=port_number option when starting the JMX Manager node.

   When the JMX Manager node boots, it starts an embedded Jetty instance and deploys the Pulse Web application at the specified or default HTTP port or 7070 by default.

   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.

2. Access the embedded Pulse application from a Web browser. If you are connected to the Tanzu 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*:*http-service-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:7070/pulse.

3. 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.

4. After you log in, Pulse displays the main cluster view for the local cluster. See Using Pulse Views.

> 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 cluster. 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

This topic explains how to host VMware Tanzu GemFire Pulse on a Web application server.

> Tanzu GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

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:

1. Set the http-service-port property to zero (-Dgemfire.http-service-port=0) when you start your Tanzu GemFire JMX Manager nodes. Setting this property to zero deactivates the embedded Web server for hosting the Pulse application.
2. Deploy the Pulse Web application to your application server. Tanzu GemFire installs the gemfire-pulse-n.n.n.war file (where n.n.n is a version number) in the tools/Pulse subdirectory of your Tanzu 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.
3. Stop the Web application server and locate the Pulse configuration in the WEB-INF/classes subdirectory.
4. Edit pulse.properties, defining or redefining any of the following configuration properties as needed for your application:

Property	Description
pulse.useLocator	Specify "true" to configure Pulse to connect to a Tanzu GemFire Locator member, or "false" to connect directly to a JMX Manager. When Pulse connects to a Tanzu GemFire locator, the locator provides the address and port of an available JMX Manager to use for monitoring the cluster. 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 cluster.
pulse.host	Specify the DNS name or IP address of the Tanzu 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 pulse.useLocator property.
pulse.port	Specify the port number of the Tanzu 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 pulse.useLocator property. If you configured pulse.useLocator=false, then pulse.port must correspond to the http-service-port setting of the JMX Manager.

For example, with the default pulse.properties configuration, Pulse connects to the locator at port 10334 and accesses any available JMX Manager:

```
pulse.useLocator=true
pulse.host=localhost
pulse.port=10334
```

With this modified configuration, Pulse accesses only the JMX Manager instance at port 8080:

```
pulse.useLocator=false
pulse.host=jmxsrv.mycluster.com
pulse.port=8080
```

5. If a JMX manager or locator is configured to use SSL, you can configure Pulse to connect to these processes. Edit pulsesecurity.properties to un-comment the standard Java SSL properties:

   ```
   javax.net.ssl.keyStore={KeyStorePath}
   javax.net.ssl.keyStorePassword={KeyStorePassword}
   javax.net.ssl.trustStore={TrustStorePath}
   javax.net.ssl.trustStorePassword={TrustStorePassword}
   ```

   Substitute the appropriate paths and passwords for the bracketed placeholders.

6. To change Pulse logging behavior, edit log4j2.xml in the same directory. The default configuration sets the logging level to INFO and specifies a message pattern that includes the date, name of the reporting logger, error level, name of the reporting thread, and the error message. Refer to the Apache Log4j documentation for details on how to specify log4j2.xml content and syntax.

7. Restart the Web application server.

8. Access the Pulse application using the address, port, and application URL that you configured in your Web application server. For example, with Tomcat the default 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.

9. 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.

10. After you log in, Pulse displays the main cluster view for the distributed system to which it has connected. See Using Pulse Views.

Configuring Pulse Authentication

This topic explains how to configure VMware Tanzu GemFire Pulse authentication. Pulse requires all users to authenticate themselves before they can use the Pulse Web application.

> VMware Tanzu GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

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 Tanzu 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.

> The credentials that you specify must have both read and write privileges in the JMX Manager node. See Configuring a JMX Manager.

Configuring Pulse to use HTTPS

You can configure Pulse to use HTTPS in either embedded or non-embedded mode.

Embedded Mode

In embedded mode, Tanzu GemFire uses an embedded Jetty server to host the Pulse Web application. To make the embedded server use HTTPS, you must enable the http SSL component in gemfire.properties or gfsecurity.properties. See SSL for details on configuring these parameters.

These SSL parameters apply to all HTTP services hosted on the JMX Manager, which includes the following:

- Developer REST API service
- Management REST API service (for remote cluster management)
- Pulse monitoring tool

When the http SSL component is enabled, all HTTP services become SSL-enabled and you must configure your client applications accordingly. For SSL-enabled Pulse, you will need to configure your browsers with proper certificates.

Non-Embedded (Standalone Web Server) Mode

In non-embedded mode where you are running Pulse on a standalone web application server, such as Tomcat, you must use the web server's SSL configuration to make the HTTP requests secure. You must also set the Spring profile.

To set the Spring profile, set the spring.profiles.active system property.

1. On a command line, run:

```
export CATALINA_OPTS=-Dspring.profiles.active=pulse.authentication.gemfire

./catalina start
```

Configuring Pulse to use Security Manager

You can configure Pulse to use the Tanzu GemFire Security Manager in either embedded or non-embedded mode.

Embedded Mode

To use Security Manager with Pulse running in embedded mode, you do not need to specify additional credentials. Pulse will automatically be configured by Tanzu GemFire to send the credentials entered in

the Pulse login page to the Security Manager for authentication and authorization.

Non-Embedded (Standalone Web Server) Mode

When running Pulse on a standalone web application server, such as Tomcat, you need to configure the Pulse Web app to use a Tanzu GemFire-specific security profile. Activate the Tanzu GemFire profile at startup with the system property:

```
-Dspring.profiles.active=pulse.authentication.gemfire
```

Configuring Pulse to use a Custom Security Profile

You can use a custom security profile only if you are NOT using a Security Manager in your cluster. A custom security profile only controls who can access the Pulse Web page. You will need to create an XML file named pulse-authentication-custom.xml that defines the credentials and roles of the users of Pulse.

Example pulse-authentication-custom.xml for Spring simple in-memory User Details Service

```
<beans:beans >
  <authentication-manager>
    <authentication-provider>
      <user-service id="userDetailsService">
        <user name="john" password="johnspassword" authorities="ROLE_CLUSTER:READ" />
        <user name="bob" password="bobspassword" authorities="ROLE_CLUSTER:READ" />
      </user-service>
    </authentication-provider>
  </authentication-manager>
</beans:beans>
```

Embedded Mode

Put pulse-authentication-custom.xml in the locator's working directory. When you start Tanzu GemFire members, specify the custom authentication profile using the -Dspring.profiles.active=pulse.authentication.custom system property. For example:

```
gfsh> start locator --name=locator --J=-Dspring.profiles.active=pulse.authentication.custom
```

Start Pulse and log in using credentials that are authorized in the custom configuration.

Non-Embedded (Standalone Web Server) Mode

Put pulse-authentication-custom.xml in the Pulse Web app's classpath, and start the app with the system property:

```
-Dspring.profiles.active=pulse.authentication.custom
```

Start Pulse and log in using credentials that are authorized in the custom configuration.

Configuring Pulse to use an OAuth Authentication Provider

You can configure Pulse to use any Authentication Provider that supports OpenID and custom scopes. You will need to:

- Register with the authentication provider, obtaining a valid client ID and client secret for Pulse.
- Configure the scopes with CLUSTER:READ, CLUSTER:WRITE, DATA:READ or DATA:WRITE.

Pulse Properties for OAuth Configuration

After you set up the authentication provider properly, create a properties file named pulse.properties (or edit it, if it already exists) to define the following required properties:

- pulse.oauth.providerId

  An ID for Pulse to use internally to identify your OAuth provider.

- pulse.oauth.providerName

  A name for Pulse to display on the login page to link to your OAuth provider.

- pulse.oauth.clientId

  The ID with which Pulse is registered with your OAuth provider.

- pulse.oauth.clientSecret

  The secret with which Pulse is registered with your OAuth provider.

- pulse.oauth.authorizationUri

  The URI for your OAuth provider's authorization endpoint.

- pulse.oauth.tokenUri

  The URI for your OAuth provider's token endpoint.

- pulse.oauth.userInfoUri

  The URI for your OAuth provider's user info endpoint.

- pulse.oauth.jwkSetUri

  The URI for your OAuth provider's JSON Web Key (JWK) Set endpoint.

- pulse.oauth.endSessionEndpoint

  The URI for your OAuth provider's endpoint to request that the end user be logged out. See the end_session_endpoint parameter of the OpenID Provider Discovery Metadata standard proposal.

- pulse.oauth.userNameAttributeName

  The attribute name used to access the user's name from your OAuth provider's user info response.

Example pulse.properties for OAuth Configuration

```
pulse.oauth.providerId=uaa
pulse.oauth.providerName=UAA
pulse.oauth.clientId=pulse
pulse.oauth.clientSecret=example-pulse-client-secret
pulse.oauth.authorizationUri=http://example.com/uaa/oauth/authorize
pulse.oauth.tokenUri=http://example.com/uaa/oauth/token
pulse.oauth.userInfoUri=http://example.com/uaa/userinfo
pulse.oauth.jwkSetUri=http://example.com/uaa/token_keys
pulse.oauth.endSessionEndpoint=http://example.com/uaa/profile
pulse.oauth.userNameAttributeName=user_name
```

Embedded Mode

1. Create the above pulse.properties file and put it in the member's working directory.
2. Start up the member with a Tanzu GemFire property (either specify it in gemfire.properties file or pass it in at startup) security-auth-token-enabled-components=pulse or security-auth-token-enabled-components=all.

   ```
   gfsh> start locator --name=locator --J=-Dgemfire.security-auth-token-enabled-components=pulse
   ```

Start pulse and you will see login page being redirected to the configured authentication provider.

Non-Embedded (Standalone Web Server) Mode

1. Edit the existing pulse.properties file and add the above content, make sure the file is in the web app's classpath
2. Start the app with the system property:

```
-Dspring.profiles.active=pulse.authentication.oauth
```

Start Pulse and you will see login page being redirected to the configured authentication provider.

Using Pulse Views

This topic explains how VMware Tanzu GemFire Pulse provides a variety of different views to help you monitor Tanzu GemFire clusters, members, and regions.

> Tanzu GemFire 10.1 deprecates Pulse in favor of VMware Tanzu GemFire Management Console. Pulse will be removed in a future release.

The following sections provide an overview of the primary Pulse views:

- Cluster View
- Member View
- Region View
- Data Browser
- Alerts Widget

Cluster View

The cluster view is a high-level overview of the cluster. 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. You can use the cluster view to drill down into details for individual members and regions in the cluster.

Use these basic controls while in Cluster view:

1. Click Members or Data to display information about Tanzu GemFire members or data regions in the cluster.
2. Click the display icons to display the Tanzu GemFire members using icon view, block view, or table view. Note that icon view is available only when displaying Members.

   For example, the following shows Tanzu GemFire Members displayed in table view:

   - While in block view or table view, click the name of a Tanzu GemFire member to display additional information in the Member View.
   - Click Topology, Server Groups, or Redundancy Zones to filter the view based on all members in the topology, configured server groups, or configured redundancy zones.

The following shows Tanzu GemFire Regions displayed in table view:

- While in block view or table view, click the name of a Tanzu GemFire region to display additional information in the Region View.

3. While in icon view, click a host machine icon to display the Tanzu GemFire members on that machine.
4. In the Alerts pane, click the severity tabs to filter the message display by the level of severity.

Cluster View Screen Components

The following table describes the data pieces displayed on the Cluster View screen.

Screen Component	Description
Cluster Status	Overall status of the cluster being monitored. Possible statuses include Normal, Warning, or Severe.
Total Heap	Total amount of memory (in GB) allocated to the Java heap across all members.
Members	Total number of members in the cluster.
Servers	Total number of servers in the cluster.
Clients	Total number of clients in the cluster.
Locators	Total number of locators in the cluster.
Regions	Total number of regions in the cluster.
Functions	Total number of functions registered in the cluster.
Unique CQs	Total number of unique CQs. Corresponds to the UNIQUE_CQ_QUERY statistic.
Subscriptions	Total number of client event subscriptions.
Cluster Members	Graphical, block, or table view of the members in the cluster.
Topology	Organizes cluster members by DistributedMember Id.
Server Groups	Organizes cluster members by server group membership. If no server groups are configured, all members appear under the "Default" server group.
Redundancy Zones	Organizes cluster members by redundancy zones. If no redundancy zones are configured, all members appear under the "Default" zone.

Screen Component	Description
Host Machine	When you mouse over a machine icon in Topology View, a pop-up appears with the following machine statistics: • *CPU Usage*. Percentage of CPU being used by Tanzu GemFire processes on the machine. • *Memory Usage*. Amount of memory (in MB) being used by Tanzu GemFire processes. • *Load Avg*. Average number of threads on the host machine that are in the run queue or are waiting for disk I/O over the last minutes. Corresponds to the Linux System statistic loadAverage1. If the load average is not available, a negative value is shown. • *Sockets*. Number of sockets currently open on the machine.
Member	When you mouse over a member icon in Graphical View, a pop-up appears with the following member statistics: • *CPU Usage*. Percentage of CPU being used by the Tanzu GemFire member process. • *Threads*. Number of threads running on the member. • *JVM Pauses*. Number of times the JVM used by the member process has paused due to garbage collection or excessive CPU usage. • *Regions*. Number of regions hosted on the member process. • *Clients*. Number of client currently connected to the member process. • *Gateway Sender*. Number of gateway senders configured on the member. • *Port*. Server port of the cache server member where clients can connect and perform cache operations. • *GemFire Version*. The version of the Tanzu GemFire member.
Member	In List View, the following data fields are displayed for each member: • *ID*. DistributedMember Id of the member. • *Name*. Name of the member. • *Host*. Hostname or IP address where the member is running. • *Heap Usage*. Amount of JVM heap memory being used by the member process. • *CPU Usage*. Percentage of CPU being used by the Tanzu GemFire member process. • *Uptime*. How long the member has been up and running. • *Clients*. Number of clients currently connected to the member. It will have a value only if the member acts as a CacheServer.
Key Statistics	Displays a few key performance measurements of the cluster (over the last 15 minutes).

Screen Component	Description
Write/Sec	Number of write operations per second that have occurred across the cluster. Each put/putAll operation counts as a write; for example, a putAll of 50 entries is counted as one write.
Read/Sec	Number of read operations per second that have occurred across the cluster.
Queries/Sec	Number of queries per second that have been executed across the cluster.
No. of JVM Pauses	Number of times the JVM has paused during the last five minutes to perform garbage collection.
WAN Information	If you have configured gateway senders or receivers for a multi-site (WAN) deployment, this box displays whether the remote cluster is reachable (working connectivity represented by a green triangle).
Disk Throughput	Total disk throughput for all disks in cluster.
Alerts View	Displays alerts for the cluster.

Member View

When you select an individual Tanzu GemFire member in Cluster View, Pulse displays the regions available on that member, as well as member-specific information such as the configured listen ports.

Use these basic controls while in Member View:

1. Click the display icons to display regions using block view or table view.
2. Use the drop down menu to select a specific member or search for specific members by name.
3. Click Cluster View to return to Cluster View. See Cluster View.
4. Click Data Browser to query region data. See Data Browser.

Member View Screen Components

The following table describes the data elements displayed on the Member View screen.

Screen Component	Description
Member Status	Overall status of the member being monitored. Possible statuses include Normal, Warning, or Severe.
Regions	Total number of regions hosted on the member.
Threads	Total number of threads being executed on the member.
Sockets	Total number of sockets currently open on the member.
Load Avg.	Average number of threads on the member that are in the run queue or are waiting for disk I/O over the last minute. Corresponds to the Linux System statistic loadAverage1. If the load average is not available, a negative value is shown.
Clients	Current number of client connections to the member.
Member Regions	Block or table view of the regions hosted on the member.
Regions	When you mouse over a region in block view, a pop-up appears with the following data fields: • *Name*. Region name. • *Type*. For example, REPLICATE, PARTITION. • *EntryCount*. Number of entries in the region. • *EntrySize*. The aggregate entry size (in bytes) of all entries. For replicated regions this field provides a value only if the eviction algorithm is set to EvictionAlgorithm#LRU_ MEMORY. All partition regions can report entry size, but note that the value includes redundant entries and also counts the size of all the secondary entries on the node.

Screen Component	Description
Regions	In table view, the following fields are listed for each region: • *Name*. Region name. • *Type*. For example, REPLICATE, PARTITION. • *EntryCount*. Number of entries in the region. • *EntrySize*. The aggregate entry size (in bytes) of all entries. For replicated regions this field provides a value only if the eviction algorithm is set to EvictionAlgorithm#LRU_ MEMORY. All partition regions can report entry size, but note that the value includes redundant entries and also counts the size of all the secondary entries on the node. • *Scope*. Scope configured for the region. • *Disk Store Name*. Name of disk stores (if any) associated with the region. • *Disk Synchronous*. True if writes to disk are set to synchronous and false if not. This field reflects the configured disk-synchronous region attribute. • *Gateway Enabled*. Whether gateway sender and receiver configurations have been defined on members hosting this region.
Member Clients	In table view, the following fields are listed for each client: • *Id*. DistributedMember ID of the client process. • *Name*. Name of the client process. • *Host*. Hostname or IP address of the client process. • *Connected*. Whether the client process is currently connected to the member. • *Queue Size*. The size of the queue used by server to send events in case of a subscription enabled client or a client that has continuous queries running on the server. • *CPU Usage*. Percentage of CPU being used by the client process. • *Uptime*. Amount of time the client process has been running. • *Threads*. Threads being used by the member clients • *Gets*. Total number of successful get requests completed. • *Puts*. Total number of successful put requests completed.
Key Statistics	Displays a few key performance measurements for the member (over the last 15 minutes).
% CPU Usage	Percentage of CPU used by the member.
Read/Sec	Number of read operations per second that have occurred on the member.
Write/Sec	Number of write operations per second that have occurred on the member. Each put/putAll operation counts as a write; for example, a putAll of 50 entries is counted as one write.

Screen Component	Description
Memory Usage	Total memory used on the member in MB.
No. of JVM Pauses	Number of times the JVM has paused during the last five minutes due to garbage collection or excessive CPU usage.
WAN Information	Displays cluster information. This dialog box only appears if you have configured WAN functionality (gateway senders and gateway receivers).
Disk Throughput	Rate of disk writes on the member.

Region View

The Pulse Region View provides a comprehensive overview of all regions in the cluster:

Use these basic controls while in Region View:

1. Click the display icons to display all members that host the region using block view or table view. Click the name of a member to change to that member's Member View.
2. Search for specific members that host the current region.
3. Hover over a member name to display information such as the region entry count, entry size, and throughput on that member.
4. Click Cluster View or Data Browser to open those screens.

Region View Screen Components

The following table describes the data elements displayed on the Region View screen.

Screen Component	Description
Region Members	Lists information about Tanzu GemFire members that host the region, either in block view or table view.
Region Member (Detail View)	When you hover over a region member in block view, a pop-up appears with the following data fields: • *Member Name*. The name of the Tanzu GemFire member hosting the region. • *EntryCount*. Number of entries for the region on that member. • *EntrySize*. The aggregate entry size (in bytes) of all entries. For replicated regions this field provides a value only if the eviction algorithm is set to EvictionAlgorithm#LRU_ MEMORY. All partition regions can report entry size, but note that the value includes redundant entries and also counts the size of all the secondary entries on the node. • *Accessor*. Indicates whether the member is an accessor member. • *Reads/Writes*. Summary of reads and writes served from memory and from disk stores over the last 15 minutes.
Region Member (Table View)	In table view, the following fields are listed for each region member: • *ID*. The unique member ID. • *Name*. Region name. • *Host*. Member hostname. • *Heap Usage*. The total amount of heap used on the member in MB. • *CPU Usage*. CPU usage as a percent of available CPU. • *Uptime*. The amount of time elapsed since the member started. • *Accessor*. Indicates whether the member is an accessor member.

Screen Component	Description
Region Detail	When you have selected a region, the right hand pane displays the following information about the region: • *Name*. Name of the region. • *Region Path*. Path for the region. • *Type*. For example, REPLICATE, PARTITION • *Members*. Number of members that are hosting the region. • *Empty Nodes*. Nodes where the region DataPolicy is defined as EMPTY or where LocalMaxMemory is set to 0. • *Entry Count*. Total number of entries in the region. • *Disk Usage*. Persistent data usage. • *Persistence*. Indicates whether the region's data is persisted to disk. • *Memory Usage*. The amount of memory used and total available memory (also shown as a percentage). • *Reads/Writes*. Summary of reads and writes served from memory and from disk stores over the last 15 minutes.

Data Browser

The Pulse Data Browser enables you to query region data. Two key attributes are available on DistributedSystemMXBean. See List of Tanzu GemFire JMX MBeans that you can use to configure limits for the result sets displayed in Data Browser:

- QueryResultSetLimit limits the number of rows that Data Browser queries return. 1000 rows are displayed by default.
- QueryCollectionsDepth limits the number of elements of a collection that Data Browser queries return. This attribute applies to query results contain collections such as Map, List, and so forth. The default value is 100 elements.

See the org.apache.geode.management.DistributedSystemMXBean JavaDocs for information about available MBean methods and attributes.

The following shows an example Data Browser view:

Use these basic controls while in Data Browser view:

1. Search for the name of a specific region.
2. Select one or more regions to display the Tanzu GemFire members that host those regions. The hosting Tanzu GemFire members appear in the Region Members section.
3. Select one or more members from the Region Members section to restrict query results to those members.
4. Type in the text of a query to execute. See Querying.
5. Display a list of previously-executed queries. Double-click on a query from the history list to copy it to the Query Editor, or delete the query from your history.
6. Execute your query or clear the contents of the Query Editor.
7. View the current query results.
8. Export the query results to a text file.
9. Return to Cluster View.

Alerts Widget

The Alerts Widget appears in the right portion of the screen and displays a list of alerts.

The alerts displayed for the cluster appear based on the alertLevel field set in the DistributedSystemMXBean. By default, log messages with the level of SEVERE are shown as alerts. You

can modify the level by using the DistributedMXBean.changeAlertLevel method. See System Alert Notifications for more information.

Use these basic controls in the Alerts Widget:

1. Select an alert level to view only alerts with a specific severity.
2. Enter text in the search box to filter the list of alerts.
3. Select an alert and click Clear to remove it from the alert list.
4. Click Clear All to remove all alerts from the widget.
5. Double-click an alert to open a pop-up window that displays the full text of the alert message.
6. Click the check mark in an alert pop-up window to acknowledge the alert. Acknowledged alerts display a check mark in the list of alerts.
7. Triple-click the alert in the pop-up or in the alert list to select the message text. You can then copy and paste the text into another application.
8. Click the X to close the pop-up alert window.

Visual Statistics Display

The Visual Statistics Display (VSD) utility reads VMware Tanzu GemFire statistics and produces graphical displays for analysis.

- VSD Overview

  VSD helps you monitor the performance of Tanzu GemFire and diagnose performance problems.

- Installing and Running VSD

  Start the VSD tool, load statistics files, and maintain the view you want on your statistics.

- Viewing Statistics in VSD

  Select statistics and view them using chart templates and customized charts.

- Quick Guide to Useful Statistics

  A large number of statistics are intended only for product support and engineering. This topic describes the most important categories and the useful statistics they contain.

- VSD Requirements

VSD Overview

This topic explains how the Visual Statistics Display (VSD) helps you monitor the performance of VMware Tanzu GemFire and diagnose performance problems.

For specific information about the statistics produced by Tanzu GemFire, see Statistics.

Your Tanzu GemFire product creates a statistical archive file named filename.gfs. The file logs useful statistics, with 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 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.

:

Installing and Running VSD

This topic explains how to install and run the Visual Statistics Display (VSD) for use with VMware Tanzu GemFire.

VSD is a free analysis tool and is provided as-is.

Start the VSD tool, load statistics files, and maintain the view that you want on your statistics.

Install VSD

Before you begin, see VSD System Requirements to view a list of platforms that are known to work with VSD.

VSD is available for download from the Broadcom Customer Support Portal. Complete the following steps:

1. Log in to the Broadcom Customer Support Portal with your customer credentials. Before downloading, ensure you meet the requirements, for more information, see the Download Broadcom products and software article.
2. Go to the VMware Tanzu GemFire downloads page. Select VMware Tanzu GemFire, click Show All Releases, and select a version.
3. Review the Terms and Conditions, and then click I agree to Terms and Conditions.
4. Click the HTTPS Download icon next to Pivotal GemFire VSD zip or Pivotal GemFire VSD for OSx to download.
5. Unpack it in a directory of your choosing.

It can be installed anywhere, but if GemFire is installed in *product-dir*, you may wish to install VSD in *product-dir*/tools/vsd so it can be conveniently launched from gfsh using the start vsd command.

For this example, assume the VSD was downloaded, unzipped, and installed in *product-dir*/tools/vsd.

> VSD is a 32-bit application. If you are running VSD on a 64-bit machine, you may need to install 32-bit OS libraries to run the application if they are not already installed. On Linux, to find out which libraries are missing, run the following ldd command:

```
ldd <product_dir>/tools/vsd/bin/vsdwishLinux
```

The VSD tool installation has two subdirectories, bin and lib:

- bin. Contains scripts and binaries that can be used to run VSD on a variety of operating systems, including:
    - vsd - script for Solaris, Linux, and Mac
    - vsd.bat - script for Windows
    - vsdwishSunOS - binary for Solaris
    - vsdwishLinux - binary for Linux
    - vsdwishDarwin - binary for Mac
    - vsdwishWindows_NT.exe - binary for Windows
- lib. The jars and binary libraries needed to run VSD.

Configure Statistics Sampling in GemFire

Before you use VSD, you must enable the collection of GemFire statistics at runtime. Set the following configurations in gemfire.properties:

```
statistic-sampling-enabled=true
statistic-archive-file=myStats.gfs
```

> The statistic-sampling-enabled property is deprecated as it will be enabled by default. It will be removed in upcoming releases.

Since collecting statistics at the default sampling rate of once every second does not affect performance, we recommend that sampling always be enabled; including during development, testing, and in production.

There is a special category of statistics called time-based statistics that can be very useful in troubleshooting and assessing the performance of some GemFire operations, but they should be used with caution because their collection can affect performance. These statistics can be enabled using the following gemfire.properties configuration:

```
enable-time-statistics=true
```

When the distributed system is up and running, every GemFire instance generates a statistics file. To simplify browsing these statistics in VSD, you may want to copy all the statistics files from all members into one directory so that you can easily load the files into 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, change directories to *product-dir*/tools/vsd/bin and enter the following command at the prompt:

- Windows:

  ```
  prompt>vsd.bat
  ```

  > To run VSD on Windows 7 or later, go to the *product-dir*/tools/vsd/bin directory. Right-click on vsd.bat and select Properties. Click Compatibility and set it to Windows XP. Repeat this step for all other executables in the bin directory.

- 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 have 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.

.gfs Time Zone Information for Matching Statistics to Log Files

When opening a .gfs file, statistics are shown in the time zone used on the *local computer* where VSD is launched. This can make it harder to relate log files to statistics if the logs are from another time zone.

To open a VSD file with the time zone used when generating it, first you need to know in which time zone the .gfs file is created. To obtain this information, use the following command:

```
strings file.gfs | head
```

For example:

```
$ strings ObjLoader?-31-03.gfs | head
Hongkong
hklp162p.oocl.com
:GemFire? x.x.x
14:46:33 PST
Linux x.x.x
```

After you obtain the time zone, modify your local computer to use the time zone used when obtaining statistics in the .gfs file. For example, on a Mac computer, you can first list available time zones:

```
sudo systemsetup -listtimezones
```

Then export the specific timezone to your environment:

```
export TZ=<timezone>
```

For example, for Hong Kong:

```
export TZ=Asia/Hong_Kong
```

Then use VSD to open the .gfs file that will now display timestamps from the original time zone.

Viewing Statistics in VSD

This topic explains how to view statistics in the Visual Statistics Display (VSD) for use with VMware Tanzu GemFire.

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 are 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 have finished, 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 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 do not 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.)

Task	Procedure
Re-read the .vsdtemplates template file into VSD after you edit 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 .vsdtemplates file. If you save the current chart as a template, you may still need to edit the .vsdtemplates 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.
Print	Print chart.
Snapshot	Write this chart as a graphic to snapshot.gif.
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.
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.
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 are currently examining.

Quick Guide to Useful Statistics

This topic provides a short guide to commonly used statistics viewable with the Visual Statistics Display (VSD) for use with VMware Tanzu GemFire.

A large number of statistics are intended only for product support and engineering. This topic describes the most important categories and the useful statistics they contain.

For a reference on GemFire statistics, see GemFire Statistics List.

Runtime Configuration

As the name implies, these statistics can help with verifying the runtime configuration of a GemFire system:

- The number of peer nodes (i.e. servers or peer accessors) in the system: *DistributionStats:nodes*. This value should be the same for every node in the system.

- The number of clients and client connections for each server: *CacheServerStats: currentClients*, and *currentClientConnections*
- The number of data entries:
    - *CachePerfStats:entries*. Each region has its own *CachePerfStats* instance per JVM named *RegionStats-REGION-NAME*, or *RegionStats-partition-REGION-NAME* for partitioned regions. Its entries statistic is the number of entries for that region in the JVM.
    - *DiskRegionStatistics* (a per region disk statistic category about the region's disk use): *entriesInVM*, and *entriesOnlyOnDisk* show the number of entries in the JVM (which can also be on disk too), and the number of entries that are only on disk, respectively.
- Partitioned Region Configuration: One of the main parameters of Partitioned Region (PR) configuration is the primary bucket distribution. To make sure that primary buckets for a PR are evenly distributed, check the *PartitionedRegionStats.primaryBucketCount* statistic for each partition. This statistic shows the number of primary buckets in a partition.

Resources

The resources that are vital for normal operation and performance are memory, file descriptors (most importantly sockets, then files), CPU, network, and disk (when disk operations, such as overflow and persistence, are involved). The following stats cover all those:

- Memory: There are several stats categories that show memory usage, for different types and granularity of memory.
    - Heap: *VMMemoryUsageStats:vmHeapMemoryStats* are all about heap usage, as are the memory stats under *VMStats:vmStats*: *freeMemory*, *totalMemory*, and *maxMemory*.
    - Non-heap memory: *VMMemoryUsageStats:vmNonHeapMemoryStats*.
    - System-wide memory stats as reported by the OS: The OS statistic category (e.g. *LinuxSystemStats* on Linux) includes various system level memory statistics, such as *freeMemory*, which shows the free memory on the host (as opposed to related to the JVM process), *physicalMemory* (total physical memory on the host), paging related statistics (*pagesSwappedIn*, *pagesSwappedOut*, *unallocatedSwap*).
    - Client and gateway queue sizes: while not actual resources, these queues may be responsible for increased memory usage, so it is good to keep them in mind when investigating memory issues. The client queue stats are in *ClientSubscriptionStats* category: *eventsQueued*, and *eventsRemoved*. The difference between the two is the current queue size. The gateway queue stats are in *GatewaySenderStatistics* category: *eventQueueSize* is the size of the queue.
- File Descriptors: file descriptor related statistics are captured in the category *VMStats: fdsOpen* and *fdLimit* show the number of open file descriptors, and the limit on file descriptors for the host, respectively.
- CPU: The CPU usage is captured in OS statistic category, e.g. *LinuxSystemStats*. The statistic *cpuActive* shows the percentage of the total available CPU time that has been used in a non-idle state.
- System load: OS statistic category (e.g. *LinuxSystemStats*) includes the *loadAverage1*, *loadAverage5*, *loadAverage15* statistics, which show the average system load for 1, 5, and 15 minutes.
- Network: OS stats also include network related stats for received (*recv*) and transmitted traffic (*recvBytes*, *xmitBytes*, *recvErrors*, *xmitErrors*).

- Disk: DiskDirStatistics:diskSpace shows the amount of disk space used for GemFire disk storage on a given disk. Above mentioned *entriesOnlyOnDisk*, and *entriesInVM* from *DiskRegionStatistics* are useful for determining the distribution of data between memory and disk, for regions that use disk overflow/persistence.

The following chart is an example of examining the *vmHeapMemoryStats* in relation to the *entriesInVM* statistic.

Throughput for Different Operations

There are several stat categories that capture the throughput for GemFire operations: *CachePerfStats* (non-PR, and PR specific), and*CacheServerStats*, which capture throughput statistics with respect to clients. Note that the PR specific instances of *CachePerfStats* cover only the specific partitioned regions, while the *CachePerfStats* instance includes aggregate stats for all non-PR regions.

- CachePerfStats category includes the following stats (all measured in the number of operations per second):
    - *gets*: the number of successful gets
    - *puts*: the number of times an entry has been added or replaced as a result of a local operation (put, create, or get which results in a load, netsearch, or netload of a value)
    - *updates*: the number of updates originating remotely
    - *putalls*: the number of putAll operations
    - *destroys*: the number of destroys
    - *Function execution: FunctionService*
    - *Queries: queryExecutions*: the number of query executions

    - *Transactions: txCommits*, *txFailures*, *txRollbacks*: the number of successful, failed, and rolled back transactions, respectively
- CacheServerStats category includes the following throughput stats for client operations on the cache server:
    - *getRequests, getResponses*
    - *getAllRequests, getAllResponses*
    - *putRequests, putResponses*
    - *putAllRequests, putAllResponses*
    - *queryRequests, queryResponses*
- Disk operations: If any disk related statistic categories are present in VSD, it means that there is disk activity (some entries are on disk). Presence of disk operations may explain a drop in throughput, as disk use slows things down.
    - *DiskRegionStatistics* (statistics about a region disk use): *writes, writeTime, writtenBytes, reads, readTime, readBytes*
    - *DiskStoreStatistics* are statistics about a specific disk store's use of disk. In addition to write/read as those in *DiskRegionStatistics*, this category includes *queueSize* statistic, which shows the current number of entries in the asynchronous queue waiting to be flushed to disk.

Reference

This topic provides links to documentation that explains VMware Tanzu GemFire properties, region attributes, the cache.xml file, cache memory requirements, and statistics.

- gemfire.properties and gfsecurity.properties (Tanzu GemFire Properties)

  You use the gemfire.properties settings to join a cluster and configure cluster member behavior. Cluster members include applications, the cache server, the locator, and other Tanzu GemFire processes.

- 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.

- Region Shortcuts

  This topic describes the various region shortcuts you can use to configure Tanzu GemFire regions.

- Exceptions and System Failures

  Your application needs to catch certain classes to handle all the exceptions and system failures thrown by VMware Tanzu GemFire.

- Memory Requirements for Cached Data

  Tanzu GemFire solutions architects need to estimate resource requirements for meeting application performance, scalability and availability goals.

- Tanzu GemFire Statistics List

  This section describes the primary statistics gathered by Tanzu GemFire when statistics are enabled.

GemFire Properties

This topic explains how to use gemfire.properties and gfsecurity.properties in VMware Tanzu GemFire to manage access to Tanzu GemFire security configuration data.

You use the gemfire.properties settings to join a cluster and configure system member behavior. Cluster members include applications, the cache server, the locator, and other Tanzu 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 also define provider-specific properties (“ssl” properties) in gfsecurity.properties instead of defining them at the command-line or in your environment.

You can specify non-ASCII text in your properties files by using Unicode escape sequences. See Using Non-ASCII Strings in VMware Tanzu GemFire Property Files for more details.

Note: Unless otherwise indicated, these settings only affect activities within this cluster - not activities between clients and servers or between a gateway sender and gateway receiver in a multi-site installation.

Setting	Definition	Applies to (Server Locator, Client)	Default
ack-severe-alert-threshold	Number of seconds the cluster will wait after the ack-wait-threshold for a message to be acknowledged before it issues an alert at severe level. A value of zero deactivates this feature.	S, L	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	S, L	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.	S, L	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.	S, L	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 async-queue-timeout or async-max-queue-size. To enable asynchronous messaging, the value must be set above zero. Valid values are in the range 0...60000. This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	S	0

Setting	Definition	Applies to (Server Locator, Client)	Default
async-max-queue-size	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 cluster. Valid values are in the range 0...1024. This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	S	8
async-queue-timeout	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 cluster. Used for handling slow receivers. This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	S, L	60000
bind-address	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 Tanzu GemFire servers to listen on, which is used unless overridden by the server-bind-address. 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. 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.	S, L	*not set*
cache-xml-file	Declarative initialization file for the member's cache.	S	cache.xml
cluster-configuration-dir	Deprecated. Use the gfsh import/export cluster-configuration command instead. This property specifies the directory in which the cluster configuration related disk-store and artifacts are stored. This property is only applicable to dedicated locators that have "enable-cluster-configuration" set to true.	L	*not set*

Setting	Definition	Applies to (Server Locator, Client)	Default
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 use the server's conflation setting (server).	S	server
conserve-sockets	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 cluster. If false, every application thread has its own sockets for distribution purposes. You can override this setting for individual threads inside your application. WAN deployments increase the messaging demands on a Tanzu GemFire system. To avoid hangs related to WAN messaging, always use the default setting of conserve-sockets=false for Tanzu GemFire members that participate in a WAN deployment.	S, L	false
delta-propagation	Specifies whether to distribute the deltas for entry updates, instead of the full values, between clients and servers and between peers.	S	true
deploy-working-dir	Working directory used when deploying JAR application files to cluster members. This directory can be local and unique to the member or a shared resource. This property will be applied only in chained classloader deployments. See Deploying Application JARs to VMware Tanzu GemFire Members for more information.	S	. (current directory)
disable-auto-reconnect	By default, a Tanzu GemFire member (both locators and servers) will attempt to reconnect and reinitialize the cache after it has been forced out of the cluster by a network partition event or has otherwise been shunned by other members. Use this property to turn off the auto-reconnect behavior. See Handling Forced Cache Disconnection Using Auto-reconnect for more details.	S, L	false
disable-jmx	By default, Tanzu GemFire automatically creates JMX MBeans. This boolean, when true, prevents creating JMX MBeans. Both gfsh and Pulse require JMX MBeans.	S, L	false
distributed-system-id	Identifier used to distinguish messages from different clusters. This is required for Portable Data eXchange (PDX) data serialization. Set distributed-system-id to different values for different systems in a multi-site (WAN) configuration, and to different values for production vs. development environments. This setting must be the same for every member of a given cluster and unique to each cluster within a WAN installation. Valid values are -1 (the default) and integers from 1 to 254 (inclusive). Values 0 and 255 have been deprecated and should not be used. -1 means no setting and indicates that WAN replication is disabled. If WAN replication is required, set distributed-system-id to a value from 1 to 254 (inclusive).	S, L	-1

Setting	Definition	Applies to (Server Locator, Client)	Default
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.	C	*not set*
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.	C	300
enable-cluster-configuration	A value of "true" causes the creation of cluster configuration on dedicated locators. The cluster configuration service on dedicated locators with this property set to "true" serves the configuration to new members joining the cluster and also saves the configuration changes caused by the gfsh commands. This property is applicable only to dedicated locators.	L	true
enable-management-rest-service	A value of "true" enables the cluster management REST service. This service requires that the cluster configuration service also be enabled. If enable-cluster-configuration is false and this is true, the management REST service would do nothing but report an error if you attempt to use it. Also, even if this property is set to true, The management REST service will be started only if http-service-port is not "0".	L	true
enable-network-partition-detection	Boolean instructing the system to detect and handle splits in the cluster, typically caused by a partitioning of the network (split brain) where the cluster is running. You must set this property to the same value across all your cluster members. In addition, this property must be set to true if you are using persistent regions and configure your regions to use DISTRIBUTED_ACK or GLOBAL scope to avoid potential data conflicts.	S, L	true
enable-time-statistics	Boolean instructing the system to track time-based statistics for the cluster and caching. Deactivated by default for performance reasons and not recommended for production environments. You must also configure statistic-sampling-enabled to true and specify a statistic-archive-file.	S, L	true
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, Tanzu 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.	S	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 Using Member Groups for more information.	S	*not set*

Setting	Definition	Applies to (Server Locator, Client)	Default
http-service-bind-address	If set, then the Tanzu GemFire member binds the embedded HTTP service to the specified address. If this property is not set but the HTTP service is enabled using http-service-port, then Tanzu GemFire binds the HTTP service to the member's local address. Used by the Tanzu GemFire Pulse Web application and the developer REST API service.	S	*not set*
http-service-port	If non-zero, then Tanzu GemFire starts an embedded HTTP service that listens on this port. The HTTP service is used to host the Tanzu GemFire Pulse Web application and the development REST API service. If you are hosting the Pulse web app on your own Web server and are not using the development REST API service, then deactivate this embedded HTTP service by setting this property to zero. Ignored if jmx-manager and start-dev-rest-api are both set to false.	S	7070
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.	S, L	false (except on locators)
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 for non-HTTP connections. Ignored if JMX Manager is false or jmx-manager-port is zero.	S, L	*not set*
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.	S, L	*not set*
jmx-manager-http-port	*Deprecated.* Use http-service-port instead.	S, L	7070
jmx-manager-port	The port this JMX Manager will listen to for client connections. If this property is set to zero then Tanzu 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.	S, L	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.	S, L	false

Setting	Definition	Applies to (Server Locator, Client)	Default
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 Tanzu GemFire Pulse.	S, L	2000
load-cluster-configuration-from-dir	Setting this property to "true" causes loading of cluster configuration from "cluster_config" directory in the locator. This property is only applicable to dedicated locators that have "enable-cluster-configuration" set to true.	L	false
locator-wait-time	The number of seconds that a member should wait for a locator to start if a locator is not available when attempting to join the cluster. Use this setting when you are starting locators and peers all at once. This timeout allows peers to wait for the locators to finish starting up before attempting to join the cluster.	S	0

locators	The list of locators used by system members. The list must be configured consistently for every member of the cluster. If the list is empty, locators are not used.
	
	For each locator, provide a host name and/or address (separated by '@', if you use both), followed by a port number in brackets. Examples:
	
	`locators=addr1[port1],addr2[port2]`
	
	`locators=host1@addr1[port1]`
	
	`locators=host1[port1],host2[port2]`
	
	On multi-homed hosts, this last notation will use the default address. If you use bind addresses for your locators, explicitly specify the addresses in the locators list—do not use just the hostname.
	
	If you specify invalid DNS hostnames in this property, any locators or servers started with gfsh will not produce log files. Make sure you provide valid DNS hostnames before starting the locator or server with gfsh.
		S, L	*not set*

lock-memory	When true, locks heap and off-heap memory into RAM to prevent the operating system from paging the memory out to disk.	S, L	false

Setting	Definition	Applies to (Server Locator, Client)	Default
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.	S, L	0
log-file	File to which a running system member writes log messages. If set to null, the default is used. Each member type has its own default output: • application: standard out • locator: <locator_name>.log • server: <server_name>.log	S, L	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 deactivated.	S, L	0
log-level	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. Valid values from lowest to highest are fine, config, info, warning, error, severe, and none.	S, L	config
max-wait-time-reconnect	Maximum number of milliseconds to wait for the cluster to reconnect on each reconnect attempt.	S, L	60000
member-timeout	Tanzu GemFire uses the member-timeout server configuration, specified in milliseconds, to detect unresponsive members. Tanzu GemFire members use a combination of heartbeat messages and TCP sockets to monitor the health of other members. If a member is suspected to be unreachable, two active checks of the member are performed. Each of those checks waits a maximum of one member-timeout. If those health checks receive a connection failure or no response within the timeout, the member is considered dead and is removed from the list of members. This setting also controls how frequently heartbeat messages are sent. They are sent at a frequency of half the member-timeout. The total time it takes to remove a member that is not responding to any network traffic is therefore 2 to 3 times the member-timeout. This is not the minimum time it takes to remove a member from the list of members in the cluster. If the Java process has crashed, but the operating system can still return a connection failure response to the health checks, the crashed member may be removed from the membership list immediately. Valid values are in the range 1000...600000.	S, L	5000 ms

Setting	Definition	Applies to (Server Locator, Client)	Default
membership-port-range	The range of ports available for membership messaging and for TCP failure detection. This is specified as two integers separated by a hyphen. Different members can use different ranges. Tanzu GemFire randomly chooses at least two unique integers from this range for the member, one for membership messaging and the other for TCP failure detection messaging. If tcp-port is configured to 0, it will also randomly select a port from this range for TCP sockets used for peer-to-peer communication only. Therefore, the specified range must include at least three available port numbers. The system uniquely identifies the member using the combined host IP address and membership port number. You may want to restrict the range of ports that Tanzu GemFire uses so the product can run in an environment where routers only allow traffic on certain ports.	S, L	41000-61000
memcached-port	If specified and is non-zero, sets the port number for an embedded Gemcached server and starts the Gemcached server.	S	0
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.	S	ASCII
name	Symbolic name used to identify this system member.	S, L	*not set*

off-heap-memory-size	Specifies the size of off-heap memory in megabytes (m) or gigabytes (g). For example:
	
	`off-heap-memory-size=4096m`
	`off-heap-memory-size=120g`
		S	*not set*

ON_DISCONNECT_CLEAR_PDXTYPEIDS	By default, the client PDX type registry will be cleared when the distributed system disconnects. To prevent the PDX type registry from being cleared, set gemfire.ON_DISCONNECT_CLEAR_PDXTYPEIDS to false.	C	*true*
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, Tanzu GemFire will not put redundant copies of data in members with the same redundancy zone setting. See Configure High Availability for a Partitioned Region for more details.	S	*not set*

Setting	Definition	Applies to (Server Locator, Client)	Default

remote-locators	Used to configure the locators that a cluster will use 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 (distributed-system-id) for the local cluster and remote locators for the remote clusters to which you will connect.
	
	For each remote locator, provide a host name, address, or both separated by @, followed by a port number in brackets. Example formats:
	
	`remote-locators=addr1[port1],addr2[port2]`
	
	`remote-locators=host1@addr1[port1]`
	
	`remote-locators=host1[port2t1],host2[port2]`
	
	`remote-locators=sni://sni_host:sni_port/locator_hostname_for_clients?dsid=5`
	
	The last example accesses a remote WAN site that has a SNI proxy as the ingress gateway. See Accessing a Site Behind a SNI Proxy for more details.
		L	*not set*

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.	S	false
security-*	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.	S, L	*not set*
security-client-accessor	Deprecated. Used for authorization. Static creation method returning an AccessControl 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.	S, L	*not set*

Setting	Definition	Applies to (Server Locator, Client)	Default
security-auth-token-enabled-components	A comma-delimited list of component names which works in conjunction with the security-manager property. If security-manager is enabled, this property determines which components will use token-based authentication instead of basic (username/password) authentication. Valid values are: "all", "management", "pulse" "all": shorthand for all the security components that support token authentication. "management": the management REST service. "pulse": the Pulse web app	L	""
security-client-accessor-pp	Deprecated. 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.	S, L	*not set*
security-client-auth-init	Used for authentication. Defines the fully-qualified static method that instantiates and returns the AuthInitialize object. AuthInitialize is responsible for producing the necessary credentials for client authentication.	S, L	*not set*
security-client-authenticator	Deprecated. Used for authentication. Static creation method returning an Authenticator object, which is used by a peer to verify the credentials of the connecting peer.	S, L	*not set*
security-client-dhalgo	Deprecated. Use ssl-enabled-components instead. 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.	S, L	*not set*
security-log-file	Used with authentication. The log file for security log messages. If not specified, the member's regular log file is used.	S, L	*not set*
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.	S, L	config
security-manager	Specifies the implementation of the SecurityManager interface that implements the callbacks that do authentication and authorization.	S, L	*not set*
security-peer-auth-init	Used for authentication. Defines the fully-qualified static method that instantiates and returns the AuthInitialize object. AuthInitialize is responsible for producing the necessary credentials for peer authentication.	S, L	*not set*
security-peer-authenticator	Deprecated. Used with authentication. Static creation method returning an Authenticator object, which is used by a peer to verify the credentials of the connecting peer.	S, L	*not set*

Setting	Definition	Applies to (Server Locator, Client)	Default
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.	S, L	1000
security-post-processor	Specifies the implementation of the PostProcessor interface that implements user-defined callbacks that can change the returned results of region get operations.	S, L	*not set*
serializable-object-filter	A semicolon-separated list of items that become full class names of objects that the system will serialize when the property validate-serializable-objects is set to true. The list is expanded using the patterns specified in the createFilter method at ObjectInputFilter.Config.	S, C	"!*"
server-bind-address	Relevant only for multi-homed hosts - machines with multiple network interface cards. Network adapter card a Tanzu 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 bind-address or, if that is not set, the machine's default card.	S	*not set*
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.	S, L	32768
socket-lease-time	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. Valid values are in the range 0 to 604800000(7 days).	S, L	60000 (1 minute)
ssl-enabled-components	Components for which to enable SSL. Comma-separated list of one or more of (cluster, gateway, jmx, locator, server, web) or "all".	C, S, L	all
ssl-endpoint-identification-enabled	Boolean. When set to true, causes clients to validate the server's hostname using the server's certificate.	C, S, L	false
ssl-use-default-context	Boolean. When set to true, allows Tanzu GemFire to use the default SSL context. When enabled, also sets ssl-endpoint-identification-enabled to true.	C, S, L	false

Setting	Definition	Applies to (Server Locator, Client)	Default
ssl-require-authentication	Boolean. Require two-way authentication for SSL-enabled components. Applies to all components except web.	S, L	true
ssl-default-alias	String. Default certificate name. If empty, use first certificate in key store.	S, L	
ssl-ciphers	Comma-separated list of SSL ciphers or "any"	S, L	any
ssl-parameter-extension	Specifies the implementation of the SSLParameterExtension interface that implements the SSL parameter extensions.	S, L	*not set*
ssl-protocols	Comma-separated list of SSL protocols or "any"	S, L	any
ssl-keystore, ssl-keystore-password	Strings. Path to key store, key store password.	S, L	
ssl-truststore, ssl-truststore-password	Strings. Path to trust store, trust store password.	S, L	
ssl-keystore-type, ssl-truststore-type	Strings. Type of key store or trust store. "JKS" indicates Java. One common alternative is "pkcs12".	S, L	JKS
standard-output-always-on	Boolean value specifying to always log to standard out even if a log file exists.	S, L	false
start-dev-rest-api	If set to true, then the developer REST API service will be started when cache is created. REST service can be configured using http-service-port and http-service-bind-address properties.	S	false

start-locator	If set, automatically starts a locator in the current process when the member connects to the cluster and stops the locator when the member disconnects.
	
	To use, specify the locator with an optional address or host specification and a required port number, in one of these formats:
	
	`start-locator=address[port1]`
	
	`start-locator=port1`
	
	If you 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 gemfire properties.
		S	*not set*

statistic-archive-file	The file to which the running system member writes statistic samples. For example: "StatisticsArchiveFile.gfs". Adding .gz suffix to the file name causes it to be compressed. To deactivate archiving set the filename to /dev/null.	S, L	*not set*
statistic-sample-rate	How often to sample statistics, in milliseconds. Valid values are in the range 1000...60000.	S, L	1000

Setting	Definition	Applies to (Server Locator, Client)	Default
statistic-sampling-enabled	Deprecated. This will be enabled by default. It will be removed in upcoming releases. Whether to collect and archive statistics on the member. 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. This setting does not apply to partitioned regions, where statistics are always enabled.	S, L	true
tcp-port	The TCP port to listen on for cache communications. If set to zero, the operating system selects an available port. Each process on a machine must have its own TCP port. Note that some operating systems restrict the range of ports usable by non-privileged users, and using restricted port numbers can cause runtime errors in Tanzu GemFire startup. Valid values are in the range 0...65535.	S, L	0
thread-monitor-enabled	Boolean. When true, enables monitoring of Tanzu GemFire-created operational threads. Informational messages are written to the log file.	S	true
thread-monitor-interval-ms	The time interval (in milliseconds) with which thread monitoring is scheduled to run.	S	60000
thread-monitor-time-limit-ms	The time period (in milliseconds) after which the monitored thread is considered to be stuck.	S	30000
use-cluster-configuration	This property is only applicable for data members (non-client and non-locator). A value of "true" causes a member to request and use the configuration from cluster configuration services running on dedicated locators. Setting this property to "false" causes a member to not request the configuration from the configuration services running on the locators.	S	true
user-command-packages	A comma separated list of Java packages that contain classes implementing the CommandMarker interface. Matching classes will be loaded when the VM starts and will be available in the GFSH command-line utility.	S	*not set*

Setting	Definition	Applies to (Server Locator, Client)	Default
validate-serializable-objects	A boolean that defaults to false. When true, instances of classes that are not internal to Tanzu GemFire and whose class name is not allowed by the list defined in the serializable-object-filter property will not be permitted to be deserialized. An IncompatibleClassException is thrown for objects not listed. JDK 8 build 121 or a later build must be installed to use this property. Servers and clients that do not meet this requirement will throw an exception upon startup.	S, C	false

- Using Non-ASCII Strings in VMware Tanzu GemFire Property Files

  You can specify Unicode (non-ASCII) characters in VMware Tanzu GemFire property files by using a \uXXXX escape sequence.

Using Non-ASCII Strings in GemFire Property Files

This topic explains how you can specify Unicode (non-ASCII) characters in VMware Tanzu 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 the following to indicate the desired property definition of log-file=myß.log.

```
log-file=my\u00df.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 about internationalization in Java, see Java Internationalization in the Oracle Java documentation.

cache.xml

This topic provides links to documentation about the VMware Tanzu GemFire cache.xml file.

Use the cache.xml file to set up general cache facilities and behavior and to create and initialize cached data regions. These topics document cache.xml requirements, provide hierarchical diagrams of <cache> and <client-cache> elements, and describe the function of each element.

> You can configure most elements of the cache.xml file and apply it to your entire cluster by using the gfsh and cluster configuration service. See gfsh Limitations for a list of items you cannot configure in gfsh and must still configure in cache.xml.

- 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> Element Hierarchy

  This section shows the hierarchy of <cache> element sub-elements that you use to configure Tanzu GemFire caches and servers.

- <cache> Element Reference

  This section documents the cache.xml sub-elements used for Tanzu GemFire server configuration. All elements are sub-elements of the <cache> element.

- <client-cache> Element Hierarchy

  This section shows the hierarchy of <client-cache> element sub-elements that you use to configure Tanzu GemFire caches and clients.

- <client-cache> Element Reference

  This section documents all cache.xml elements that you use to configure Tanzu GemFire clients. All elements are sub-elements of the <client-cache> element.

cache.xml Quick Reference

This topic documents VMware Tanzu GemFire cache.xml file requirements and variables. It also links to specific element sections for server, client, and WAN configuration.

- Cache XML Requirements
- Variables in cache.xml
- Configuration Quick Reference

Cache XML Requirements

The cache.xml file has these requirements:

- The contents must conform to the XML schema definition provided in cache-1.0.xsd. The schema definition file is available at http://geode.apache.org/schema/cache/cache-1.0.xsd.
- The file must include a <cache> schema declaration of one of the following forms:
  - Server or peer cache:

    ```
    <?xml version="1.0" encoding="UTF-8"?>
    <cache
       xmlns="http://geode.apache.org/schema/cache"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cac
    he-1.0.xsd"
       version="1.0">
    ...
    </cache>
    ```

  - Client cache:

```
<?xml version="1.0" encoding="UTF-8"?>
<client-cache
   xmlns="http://geode.apache.org/schema/cache"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cac
he-1.0.xsd"
   version="1.0">
...
</client-cache>
```

- Any class name specified in the file must have a public zero-argument constructor and must implement the org.apache.geode.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:

```
<?xml version="1.0" encoding="UTF-8"?>
<cache
   xmlns="http://geode.apache.org/schema/cache"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/schema/cache/cache-1.0.xsd"
   version="1.0">
  <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
  - <cache-server>
  - <region>
  - <region-attributes>

  You can set the same server configuration properties using the org.apache.geode.cache.server.CacheServer and org.apache.geode.cache.Cache interfaces. For detailed information, see the online Java API documentation.
- Client Configuration
  - <client-cache> Element Reference
  - <pool>
  - <region>

You can set the same client configuration properties using the org.apache.geode.cache.clientClientCache and Pool interfaces. For detailed information, see the online Java API documentation.

- Multi-site (WAN) Configuration and Asynchronous Event Queue Configuration
  - <gateway-sender>
  - <gateway-receiver>
  - <async-event-queue>

  The gateway sender and receiver APIs in org.apache.geode.cache.util provide corresponding getter and setter methods for these attributes.

Element Hierarchy

This topic shows the hierarchy of <cache> element sub-elements that you use to configure VMware Tanzu GemFire caches and servers.

For details, see <cache> Element Reference.

```
<cache>
  <cache-transaction-manager>
    <transaction-listeners>
    <transaction-writer>
  <gateway-hub>
    <gateway>
      <gateway-endpoint>
      <gateway-listener>
      <gateway-queue>
  <gateway-sender>
    <gateway-event-filter>
  <gateway-receiver>
  <gateway-conflict-resolver>
    <async-event-queue>
    <async-event-listener>
      <class-name>
      <parameter>
        <string>
        <declarable>
  <cache-server>
    <group>
    <client-subscription>
    <custom-load-probe>
       <class-name>
       <parameter>
         <string>
         <declarable>
  <pool>
    <locator>
    <server>
  <disk-store>
    <disk-dirs>
      <disk-dir>
  <pdx>
    <pdx-serializer>
      <class-name>
      <parameter>
        <string>
        <declarable>
  <region-attributes>
    <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>
  <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>
  <compressor>
    <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>
  <jndi-binding>
    <config-property>
      <config-property-name>
      <config-property-type>
      <config-property-value>
<region>
  <region-attributes>
  <index>
  <lucene:index>
```

```
      <field>
    <entry>
      <key>
        <string>
        <declarable>
      <value>
        <string>
        <declarable>
  <vm-root-region>
  <function-service>
    <function>
      <class-name>
      <parameter>
        <string>
        <declarable>
  <resource-manager>
  <serialization-registration>
    <serializer>
      <class-name>
        <instantiator>
      <class-name>
        <backup>
  <initializer><initializer>
    <class-name>
    <parameter>
      <string>
      <declarable>
</cache>
```

Element Reference

This topic documents the cache.xml sub-elements used for VMware Tanzu GemFire server configuration. All elements are sub-elements of the <cache> element.

For Tanzu GemFire client configuration, see <client-cache> Element Reference.

API: org.apache.geode.cache.CacheFactory

cache Attributes

cache 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

Attribute	Description	Default
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
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 acknowledged, queued events that have already been processed by the clients. Setting a lower frequency can result in an increase in duplicated events sent to the clients after failover. Does not affect the failover timing or queue consistency.	1
search-timeout	Time, in seconds, that a netSearch operation can wait for data before timing out. Adjust this value 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

<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>
```

---

transaction-listener

<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>.

---

transaction-writer

<transaction-writer>

When you commit a transaction, a TransactionWriter can perform additional tasks, including cancelling the transaction.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See <class-name> and <parameter>.

---

dynamic-region-factory

<dynamic-region-factory>

Deprecated. 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 org.apache.geode.cache.DynamicRegionFactory class in your Java code to dynamically create regions.

You cannot use this element to dynamically create *partitioned* regions.

> Use of the DynamicRegionFactory class and the <dynamic-region-factory> element are deprecated in favor of the org.apache.geode.cache.execute.FunctionService class and the <function-service> element.

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: org.apache.geode.cache.DynamicRegionFactory

<dynamic-region-factory> Attributes

Attribute	Description	Default
disable-persist-backup	When set to false, the factory is persisted on disk.	false
disable-register-interest	When set to false, client regions created by the factory register interest in all keys in a corresponding server cache region.	false
pool-name	The name of a connection pool used by the client factory to communicate with the server-side factory.	none

Example:

```
<dynamic-region-factory
  pool-name=myPool>
  <disk-dir>/home/gemfire/myDiskdir</disk-dir>
</dynamic-region-factory>
```

disk-dir

<disk-dir>

Specifies a region or disk store's disk directory.

<disk-dir> 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-sender

<gateway-sender>

Configures a gateway sender to distribute region events to another Tanzu GemFire site. See Configuring a Multi-site (WAN) System.

API: GatewaySender

<gateway-sender> Attributes

Attribute	Description	Default
parallel	Value of true or false that specifies the type of gateway sender that Tanzu GemFire creates.	false

Attribute	Description	Default
dispatcher-threads	Number of dispatcher threads that are used to process region events from a gateway sender queue or asynchronous event queue.	5
order-policy	When the dispatcher-threads attribute is greater than 1, order-policy configures the way in which multiple dispatcher threads process region events from a serial gateway queue or serial asynchronous event queue. This attribute can have one of the following values: • 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. 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 Tanzu 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. You cannot configure the order-policy for a parallel event queue, because parallel queues cannot preserve event ordering for regions. Only the ordering of events for a given partition (or in a given queue of a distributed region) can be preserved.	key
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 Tanzu GemFire cluster to which this gateway sender will send region events. This value corresponds to the distributed-system-id property specified in locators for the remote cluster. This attribute is required.	null
manual-start	Deprecated. Boolean value that specifies whether you need to manually start the gateway sender. If you supply a null value, the default value of false is used, and the gateway sender attempts to start automatically. *A manual start is likely to cause data loss, so manual start should never be used in a production system.*	false
socket-buffer-size	Size of the socket buffer that sends messages to remote sites. This size should match the size of the socket-buffer-size attribute of remote gateway receivers that process region events.	32768
socket-read-timeout	Amount of time in milliseconds that the gateway sender will wait to receive an acknowledgment from a remote site. By default this is set to 0, which means there is no timeout. If you set this timeout, you must set it to a minimum of 30000 (milliseconds). Setting it to a number lower than 30000 will generate an error message and reset the value to the default 0.	0

Attribute	Description	Default
enable-batch-conflation	Boolean value that determines whether Tanzu GemFire should conflate messages.	false
batch-size	Maximum number of messages that a batch can contain.	100
batch-time-interval	Maximum amount of time, in ms, that can elapse before a batch is delivered, when no events are found in the queue to reach the batch-size.	1000
enable-persistence	Boolean value that determines whether Tanzu 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, Tanzu 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 Tanzu GemFire logs an alert.	0

Example:

```
<cache>
 ...
 <gateway-sender
  id="remoteA"
  parallel="true"
  remote-distributed-system-id="1">
  <gateway-event-filter>
   <class-name>org.apache.geode.util.SampleEventFilter</class-name>
   <parameter
    name="param1">
    <string>"value1"</string>
   </parameter>
  </gateway-event-filter>
 </gateway-sender>
 ...
</cache>
```

gateway-event-filter

<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>.

Example:

```
<gateway-event-filter>
  <class-name>org.apache.geode.util.SampleEventFilter</class-name>
  <parameter name="param1">
    <string>"value1"</string>
  </parameter>
</gateway-event-filter>
```

gateway-event-substitution-filter

<gateway-event-substitution-filter>

A GatewayEventSubstitutionFilter provides a way to specify a substitute value to be stored in the GatewayQueueEvent and enqueued in the RegionQueue. You can optionally add one GatewayEventSubstitutionFilter implementation to a gateway sender.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See <class-name> and <parameter>.

Example:

```
<gateway-event-substitution-filter>
  <class-name>org.apache.geode.util.SampleEventSubstitutionFilter</class-name>
  <parameter name="param1">
    <string>"value1"</string>
  </parameter>
</gateway-event-substitution-filter>
```

gateway-receiver

<gateway-receiver>

Configures a gateway receiver to receive and apply region events that were distributed from another Tanzu GemFire site. You can only specify one gateway receiver on a member. See Configuring a Multi-site (WAN) System.

API: GatewayReceiverFactory

<gateway-receiver> 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 connects to gateway senders in other sites. Tanzu 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 STARTPORT value is inclusive while the ENDPORT value is exclusive. For example, if you specify STARTPORT="50510" and ENDPOINT="50520", Tanzu GemFire chooses a port value from 50510 to 50519.	5000

Attribute	Description	Default
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. Tanzu 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", Tanzu 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.	
hostname-for-senders	Attribute where you can specify an IP address or hostname for gateway sender connections. If you configure hostname-for-senders, locators will use the provided hostname or IP address when instructing gateway senders on how to connect to gateway receivers. If you provide "" or null as the value, by default the gateway receiver's bind-address will be sent to clients.	
manual-start	When set to false, the gateway receiver will automatically start when the receiver is created. If set to true, you must manually start the receiver.	true
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 socket-buffer-size setting of gateway senders that connect to this receiver.	32768

Example:

```
<cache>
  <gateway-receiver start-port="1530" end-port="1551">
  </gateway-receiver>
</cache>
```

gateway-conflict-resolver

<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 cluster (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

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>.

API: org.apache.geode.cache.util.GatewayConflictResolver

Example:

```
<gateway-conflict-resolver>
  <class-name>
    myPackage.MyConflictResolver
  </class-name>
</gateway-conflict-resolver>
```

---

async-event-queue

<async-event-queue>

Configures a queue for sending region events to an AsyncEventListener implementation (for example, for write-behind event handling).

API: org.apache.geode.cache.asyncqueue.AsyncEventQueue

<async-event-queue> Attributes

Attribute	Description	Default
id	Unique identifier for the queue. This attribute is required.	null
parallel	Boolean value that specifies the type of queue that Tanzu GemFire creates.	false
batch-size	Maximum number of messages that a batch can contain.	100
batch-time-interval	Maximum amount of time, in ms, that can elapse before a batch is delivered, when no events are found in the queue to reach the batch-size.	5
enable-batch-conflation	Boolean value that determines whether Tanzu GemFire should conflate messages.	false
disk-store-name	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, Tanzu GemFire uses the default disk store for overflow and queue persistence.	null specifies the default disk store
disk-synchronous	For regions that write to disk, boolean that specifies whether disk writes are done synchronously for the region.	true
dispatcher-threads	Number of dispatcher threads that are used to process region events from the queue.	5
forward-expiration-destroy	When true, forwards expiration destroy operations to AsyncEventListener.	false
maximum-queue-memory	Maximum amount of memory in megabytes that the queue can consume before overflowing to disk.	100 MB

Attribute	Description	Default
order-policy	When the dispatcher-threads attribute is greater than 1, order-policy configures the way in which multiple dispatcher threads process region events from the queue. This attribute can have one of the following values: • 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. 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 Tanzu 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. You cannot configure the order-policy for a parallel event queue, because parallel queues cannot preserve event ordering for regions. Only the ordering of events for a given partition (or in a given queue of a distributed region) can be preserved.	key
pause-event-processing	When true, event dispatching from the queue to the listeners are paused when the AsyncEventQueue is started.	false
persistent	Boolean value that determines whether Tanzu 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

<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>.

API: org.apache.geode.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

<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 org.apache.geode.cache.server.CacheServer and org.apache.geode.cache.Cache interfaces. For detailed information, see the online Java API documentation.

API: org.apache.geode.cache.server.CacheServer

<cache-server> Attributes

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 cannot translate into an IP address, the client will have no route to the server's host and will not 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 bind-address setting is used.	null
load-poll-interval	Frequency, in milliseconds, to poll the load probe for load information on the server.	5000 (5 seconds)

Attribute	Description	Default Value
max-connections	Maximum number of client connections for the server. When the maximum is reached, the server refuses additional client connections. Note: Set this at least as high as max-threads.	1200
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 0 (zero) setting causes the server to use a thread for every client connection. Note: Set this to a value less than or equal to the value of max-connections.	0
maximum-message-count	Maximum number of messages allowed in a subscription queue. When the queue reaches this limit, messages block. Note: Used only if client-subscription 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. Note: A setting of 0 or a negative number deactivates client health monitoring. VMware recommends that you do not accidentally deactivate health monitoring.	60000 (1 minute)
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. Note: 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
tcp-no-delay	When set to true, enables TCP_NODELAY for Tanzu GemFire server connections to clients.	false

Example:

```
<cache>
  <cache-server
    port="40404"/>
  ...
</cache>
```

client-subscription

<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: org.apache.geode.cache.server.ClientSubscriptionConfig

<client-subscription> 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 null. Specified in megabytes for mem and as a positive integer for entry.	1
disk-store-name	Used if eviction-policy is not null. 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

Example:

```
<cache>
  <cache-server port=4444>
    <client-subscription eviction-policy="entry | mem" capacity=35 overflow-directory="OverflowDir">
    </client-subscription>
    ...
  </cache-server>
</cache>
```

---

custom-load-probe

<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>.

Default: If this is not defined, the default Tanzu GemFire load probe is used.

API: org.apache.geode.cache.server.setLoadProbe

Example:

```
<custom-load-probe>
  <class-name>
    myPackage.MyLoadProbe
  </class-name>
</custom-load-probe>
```

---

pool

<pool>

Use for client caches. Defines a client's server pool used to communicate with servers running in a different cluster.

API: org.apache.geode.cache.client.PoolFactory

<pool> Attributes

Attribute	Description	Default
free-connection-timeout	Amount of time, in milliseconds, a thread will wait to get a pool connection before timing out with an exception. This timeout keeps threads from waiting indefinitely when a pool's max-connections has been reached and all connections in the pool are in use by other threads. Takes effect only when pr-single-hop-enabled is false. When pr-single-hop-enabled is true, use server-connection-timeout.	10000 (10 seconds)
idle-timeout	Maximum time, in milliseconds, a pool connection can stay open without being used when there are more than min-connections in the pool. Pings over the connection do not count as connection use. If set to -1, there is no idle timeout. If load-conditioning-interval is set to a nonzero value, then idle-timeout cannot exceed that value. In this case, if idle-timeout is set to -1 or to a value greater than the load-conditioning-interval, it is reset to the same value as load-conditioning-interval.	120000 (2 minutes)
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. VMware recommends that you set load-polling-interval higher than the server's load-poll-interval and the client's ping-interval.	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 free-connection-timeout is reached. If set to -1, there is no maximum. The setting must indicate a cap greater than min-connections. Takes effect only when pr-single-hop-enabled is false. When pr-single-hop-enabled is true, use max-connections-per-server.	-1
max-connections-per-server	Maximum number of connections pooled to a single server. If the maximum connections are in use, an operation requiring a client-to-server connection to that server blocks until a connection becomes available or server-connection-timeout is reached. If set to -1, there is no maximum. The setting must indicate a cap greater than min-connections-per-server. Takes effect only when pr-single-hop-enabled is true. When pr-single-hop-enabled is false, use max-connections.	-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 max-connection setting. The setting must be an integer greater than or equal to 0. Takes effect only when pr-single-hop-enabled is false. When pr-single-hop-enabled is true, use min-connections-per-server.	1

Attribute	Description	Default
min-connections-per-server	Minimum number of connections for each cache server in the cluster. The system tries to keep available at least this many connections to each server at all times. When a new pool is created, it immediately begins creating connections to each server, up to this minimum. 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 max-connections-per-server setting. The setting must be an integer greater than or equal to 0. Takes effect only when pr-single-hop-enabled is true. When pr-single-hop-enabled is false, use min-connections.	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 RegionService 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. Note: 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. Note: Set this to a value lower than the server's maximum-time-between-pings.	10000
pr-single-hop-enabled	Optimizes access to partitioned region data in the servers. When true, the pool configuration is also affected by the attributes max-connections-per-server, min-connections-per-server, and server-connection-timeout. When false, the pool configuration is also affected by the attributes max-connections, min-connections, and free-connection-timeout. For details about the way these settings interact, see Configuring a Client/Server System.	true
read-timeout	Maximum time, in milliseconds, for the client to wait for a response from a server.	10000 (10 seconds)
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

Attribute	Description	Default
server-connection-timeout	Amount of time, in milliseconds, a thread will wait to get a pool connection to a specific server, before timing out with an exception. This timeout keeps threads from waiting indefinitely when the pool's max-connections-per-server has been reached and all connections in a pool are in use by other threads. Takes effect only when pr-single-hop-enabled is true. When pr-single-hop-enabled is false, use free-connection-timeout.	0
server-group	Logical named server group to use from the pool. A null value uses the global server group to which all servers belong. Note: This is only used when the locator 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. Note: Used only when subscription-redundancy is not 0 (zero).	100
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, no servers are used. If set to -1, all available servers are used.	0

Example:

```
<pool
 name="publisher"
 subscription-enabled="true">
 <locator
  host="myLocatorAddress1"
  port="12345"/>
 <locator
  host="myLocatorAddress2"
  port="45678"/>
</pool>
```

locator

<locator>

Addresses and ports of the locators to connect to. You can define multiple locators for the pool.

> Provide a locator list or server list, but not both.

<locator> Attributes

Attribute	Description	Default
host	Hostname of the locator	
port	Port number of the locator	

Example:

```
<pool ...>
  <locator
    host="myLocatorHost"
    port="12345"/>
</pool>
```

---

server

<server>

Addresses and ports of the servers to connect to.

> Provide a server list or locator list, but not both.

Default:

<server> 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

<disk-store>

Defines a pool of one or more disk stores, which can be used by regions, and client subscription queues.

Default: The cache default disk store, named “DEFAULT”, is used when disk is used but no disk store is named.

API: org.apache.geode.cache.DiskStore

<disk-store> Attributes

Table 10. <disk-store> 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 operation log (oplog) will become able to be compacted. Until the oplog 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 operation log (oplog) 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.	
disk-usage-warning-percentage	Disk usage above this threshold generates a warning message. For example, if the threshold is set to 90%, then on a 1 TB drive falling under 100 GB of free disk space generates the warning. Set to 0 (zero) to deactivate.	90
disk-usage-critical-percentage	Disk usage above this threshold generates an error message and shuts down the member's cache. For example, if the threshold is set to 99%, then falling under 10 GB of free disk space on a 1 TB drive generates the error and shuts down the cache. Set to 0 (zero) to deactivate.	99

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

<disk-dirs>

An element of a disk store that defines a set of <disk-dir> elements.

disk-dir

<disk-dir>

Specifies a region or disk store's disk directory.

<disk-dir> 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

<pdx>

Specifies the configuration for the Portable Data eXchange (PDX) method of serialization.

API: org.apache.geode.cache.CacheFactory.setPdxReadSerialized, setPdxDiskStore, setPdxPersistent, setPdxIgnoreUnreadFields and org.apache.geode.cache.ClientCacheFactory.setPdxReadSerialized, setPdxDiskStore, setPdxPersistent, setPdxIgnoreUnreadFields

<pdx> Attributes

Attribute	Description	Default
read-serialized	Set to true to have PDX deserialization produce a PdxInstance instead of an instance of the domain class.	
ignore-unread-fields	Set 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. 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>
    org.apache.geode.pdx.ReflectionBasedAutoSerializer
   </class-name>
   <parameter name="classes">
    <string>com.company.domain.DomainObject</string>>
   </parameter>
  </pdx-serializer>
 </pdx>
 ...
</cache>
```

---

pdx-serializer

<pdx-serializer>

Allows you to configure the PdxSerializer for this Tanzu GemFire member.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See <class-name> and <parameter>.

Default:

API: org.apache.geode.cache.CacheFactory.setPdxSerializer

Example:

```
<cache>
 <pdx>
  <pdx-serializer>
   <class-name>com.company.ExamplePdxSerializer</class-name>
  </pdx-serializer>
 </pdx>
 ...
</cache>
```

---

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: org.apache.geode.cache.RegionFactory or org.apache.geode.cache.ClientRegionFactory

<region-attributes> Attributes

Attribute	Description	Default

concurrency-level	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 Tanzu 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.
	
	Note: Before you modify this, read the concurrency level description, then see the Java API documentation for java.util.ConcurrentHashMap.
	
	API: setConcurrencyLevel
	
	Example:
	
	`<region-attributes`
	`  concurrency-level="10">`
	`</region-attributes>`
		16 (threads)

Attribute	Description	Default

data-policy	Specifies how the local cache handles data for a region. This setting controls behavior such as local data storage and region initialization.
	
	Note: 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.
	
	You can specify the following data policies:
	
	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 subscription-attributes interest-policy to all.
	
	normal: Data used locally (accessed with get, stored with put, etc.) is stored in the local cache. This policy allows the contents in the cache to differ from other caches.
	
	partition: Data is partitioned across local and remote caches using the automatic data distribution behavior of partitioned regions. Additional configuration is done in the partition-attributes.
	
	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 scope, behaving the same as for normal.
	
	persistent-partition: Behaves the same as partition and also persists data to disk.
	
	persistent-replicate: Behaves the same as replicate and also persists data to disk.
	
	preloaded: Initializes like a replicated region, then, once initialized, behaves like a normal region.
	
	API: setDataPolicy
	
	Example:
	
	`<region-attributes data-policy="replicate">`
	`</region-attributes>`
	
	This is similar to using a region shortcut withrefid, however when you use the REPLICATE region shortcut, it automatically sets the region's scope to distributed-ack.
		normal

Attribute	Description	Default
	<region-attributes refid="REPLICATE"> </region-attributes> If you use data-policy, you must set the scope explicitly.	
enable-async-conflation	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 deactivate 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. Note: To use this attribute, asynchronous queues must be enabled for slow consumers, specified with the async* gemfire properties. API: setEnableAsyncConflation Example: <region-attributes enable-async-conflation="false"> </region-attributes>	true
enable-subscription-conflation	Boolean for server regions that specifies whether the server can conflate its messages to the client. A true value enables conflation. Note: The client can override this setting with the conflate-events property in its gemfire.properties. API: setEnableSubscriptionConflation Example: <region-attributes enable-subscription-conflation="true"> </region-attributes>	false
gateway-sender-ids	Specifies one or more gateway sender IDs to use for distributing region events to remote Tanzu GemFire sites. Specify multiple IDs as a comma-separated list. API: addGatewaySenderId Example: <region-attributes gateway-sender-ids="nwsender,swsender"> </region-attributes>	not set

Attribute	Description	Default

async-event-queue-ids	Specifies one or more asynchronous event queues to use for distributing region events an AsyncEventListener implementation (for example, for write-behind cache event handling). Specify multiple IDs as a comma-separated list.
	
	API: addAsyncEventQueueId
	
	Example:
	
	`<region-attributes`
	`  async-event-queue-ids="customerqueue,ordersqueue">`
	`</region-attributes>`
		not set

hub-id	If the enable-gateway attribute is set to true, a comma-separated list of gateway hub IDs that receive events from the region. Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the gateway-sender-id attribute of the <region-attributes> element.	null

id	Stores the region attribute settings in the cache with this identifier. Once stored, the attributes can be retrieved using the region attribute refid.
	
	API: setId
	
	Example:
	
	`<region-attributes`
	`  id="persistent-replicated">`
	`</region-attributes>`
		not set

ignore-jta	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.
	
	API: setIgnoreJTA
	
	Example:
	
	`<region-attributes`
	`  ignore-jta="true">`
	`</region-attributes>`
		false

index-update-type	Specifies whether region indexes are maintained synchronously with region modifications, or asynchronously in a background thread. In the cache.xml file, this is set as a value, asynchronous or synchronous, assigned to the index-update-type region attribute. Set this through the API by passing a boolean to the setIndexMaintenanceSynchronous method.
	
	API: setIndexMaintenanceSynchronous
	
	Example:
	
	`<region-attributes`
	`  index-update-type="asynchronous">`
	`</region-attributes>`
		synchronous updates

Attribute	Description	Default

initial-capacity	Together with the load-factor region attribute, sets the initial parameters on the underlying java.util.ConcurrentHashMap used for storing region entries.
	
	API: setInitialCapacity
	
	Example:
	
	`<region-attributes`
	`  initial-capacity="20">`
	`</region-attributes>`
		16

is-lock-grantor	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 is-lock-grantor set to true. This attribute is only relevant for regions with global scope, as only they allow locking. It affects implicit and explicit locking.
	
	API: setLockGrantor
	
	Example:
	
	`<region-attributes`
	`  is-lock-grantor="true">`
	`</region-attributes>`
		false

load-factor	Together with the initial-capacity region attribute, sets the initial parameters on the underlying java.util.ConcurrentHashMap used for storing region entries. This must be a floating point number between 0 and 1, inclusive.
	
	Note:
	
	Before you set this attribute, read the discussion of initial capacity and load factor, then see the Java API documentation for java.util.ConcurrentHashMap.
	
	API: setLoadFactor
	
	Example:
	
	`<region-attributes`
	`  load-factor="0.85">`
	`</region-attributes>`
		.75

mirror-type	Deprecated	

Attribute	Description	Default

pool-name	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.
	
	API: setPoolName
	
	Examples:
	
	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:
	
	`<client-cache>`
	`  <pool name="DatabasePool"`
	`  subscription-enabled="true">`
	`  ...`
	`  </pool>`
	`  <pool>`
	`  name="OtherPool"`
	`  subscription-enabled="true">`
	`  ...`
	`  </pool>`
	`  <region ... >`
	`  <region-attributes`
	`  pool-name="DatabasePool">`
	`  </region-attributes>`
	`  ...`
	
	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:
	
	`<client-cache>`
	`  <pool`
	`  name="publisher"`
	`  subscription-enabled="true">`
	`  ...`
	`  </pool>`
	`  <region ...>`
	`  name="myRegion"`
	`  refid="CACHING_PROXY">`
	`  </region>`
	`</client-cache>`
		not set

Attribute	Description	Default

disk-store-name	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 disk-synchronous region attribute specifies whether writes are done synchronously or asynchronously.
	API: setDiskStoreName
	Example:
	`<region-attributes`
	`  disk-store-name="myStoreA" >`
	`</region-attributes>`
		null

disk-synchronous	For regions that write to disk, boolean that specifies whether disk writes are done synchronously for the region.
	API: setDiskSynchronous
	Example:
	`<region-attributes`
	`  disk-store-name="myStoreA"`
	`  disk-synchronous="true">`
	`</region-attributes>`
		true

refid	Retrieves region shortcuts and user-defined named region attributes for attributes initialization
	API: setRefId
	Example:
	`<region-attributes`
	`  refid="persistent-replicated">`
	`  <!-- Override any stored`
	`  attribute settings that you`
	`  need to ... -->`
	`</region-attributes>`
		not set

Attribute	Description	Default

scope	Determines how updates to region entries are distributed to the other caches in the cluster 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.
	
	You can configure the most common of these options with Tanzu GemFire region shortcuts in RegionShortcut and ClientRegionShortcut.
	
	Server regions that are not partitioned must be replicated with distributed-ack or global scope. The region shortcuts that specify REPLICATE have distributed-ack scope.
	
	Set one of the following scope values:
	
	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 cluster.
	
	global — Events are distributed to remote caches with global locking to ensure distributed cache consistency.
	
	API: setScope
	
	Example:
	
	`<region-attributes scope="distributed-ack">`
	`</region-attributes>`
		distributed-no-ack

statistics-enabled	Boolean specifying whether to gather statistics on the region. Must be true to use expiration on the region. Tanzu GemFire provides a standard set of statistics for cached regions and region entries, which give you information for fine-tuning your cluster. Unlike other Tanzu GemFire statistics, statistics for local and distributed regions are not archived and cannot be charted. They are kept in instances of org.apache.geode.cache.CacheStatistics and made available through the region and its entries through the Region.getStatistics and Region.Entry.getStatistics methods.
	
	API: setStatisticsEnabled
	
	Example:
	
	`<region-attributes statistics-enabled="true">`
	`</region-attributes>`
		false

Attribute	Description	Default

cloning-enabled	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.
	
	API: setCloningEnabled
	
	Example:
	
	`<region-attributes`
	`  cloning-enabled="true">`
	`</region-attributes>`
		false

concurrency-checks-enabled	Determines whether members perform checks to provide consistent handling for concurrent or out-of-order updates to distributed regions. For more information, see Consistency for Region Updates.
	
	Note: Applications that use a client-cache may want to deactivate concurrency checking to see all events for a region. Tanzu 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 unsynchronized with the server cache.
	
	API: setConcurrencyChecksEnabled
	
	Example:
	
	`<region-attributes`
	`  concurrency-checks-enabled="true">`
	`</region-attributes>`
		true

off-heap	Specifies that the region uses off-heap memory to store entry values, including values for region entries and queue entries. The region will still use heap memory for everything else, such as entry keys and the ConcurrentHashMap.
	
	API: setOffHeap
	
	Example:
	
	`<region-attributes`
	`  off-heap="true">`
	`</region-attributes>`
		false

key-constraint

<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.

> Set the constraint in every cache where you create or update the region entries. For client/server installations, match constraints between client and server and between clusters. 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: org.apache.geode.cache.RegionFactory.setKeyConstraint

Example:

```
<region-attributes>
  <key-constraint>
    java.lang.String
  </key-constraint>
</region-attributes>
```

---

value-constraint

<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 is not 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.

> Set the constraint in every cache where you create or update the region entries. For client/server installations, match constraints between client and server and between clusters. 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: org.apache.geode.cache.RegionFactory.setValueConstraint

Example:

```
<region-attributes>
  <value-constraint>
    cacheRunner.Portfolio
  </value-constraint>
</region-attributes>
```

---

region-time-to-live

<region-time-to-live>

Deprecated. 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: org.apache.geode.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

<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 cluster. Within the region-time-to-live or region-idle-time element, this element specifies the expiration rules for the entire region. The elements region-time-to-live or region-idle-time are deprecated.

API: See APIs for entry-time-to-live, entry-idle-time, region-time-to-live, region-idle-time

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to 0 (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 cluster. 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 cluster. 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">
  <region-time-to-live>
    <expiration-attributes
      timeout="60"
      action="local-destroy"/>
  </region-time-to-live>
</region-attributes>
```

---

custom-expiry

<custom-expiry>

Specifies the custom class that implements org.apache.geode.cache.CustomExpiry. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements.

API: org.apache.geode.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.

---

region-idle-time

<region-idle-time>

Deprecated. Expiration setting that specifies how long the region can remain in the cache without anyone accessing it.

Default: not set - no expiration of this type

API: org.apache.geode.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

<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 cluster. Within the region-time-to-live or region-idle-time element, this element specifies the expiration rules for the entire region. The elements region-time-to-live or region-idle-time are deprecated.

API: See APIs for entry-time-to-live, entry-idle-time, region-time-to-live, region-idle-time

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to 0 (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 cluster. 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 cluster. 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">
 <region-idle-time>
  <expiration-attributes
   timeout="60"
   action="local-destroy"/>
 </region-idle-time>
</region-attributes>
```

custom-expiry

<custom-expiry>

Specifies the custom class that implements org.apache.geode.cache.CustomExpiry. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements.

API: org.apache.geode.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.

---

entry-time-to-live

<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.

An entry's expiration based on its Time to Live (TTL) can be calculated from either its creation time or its last modification time. Use the reset-policy attribute on the entry-time-to-live element to control this:

- never: TTL begins at entry creation.
- last-modified: TTL resets whenever the entry is updated (default).

> reset-policy is only supported on cache servers.

Default: not set - no expiration of this type.

API: org.apache.geode.cache.RegionFactory.setEntryTimeToLive

Example:

```
<region-attributes
  statistics-enabled="true">
  <entry-time-to-live reset-policy="last-modified"
    <expiration-attributes
      timeout="60"
      action="local-destroy"/>
  </entry-time-to-live>
</region-attributes>
```

---

expiration-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 cluster. Within the region-time-to-live or region-idle-time element, this element specifies the expiration rules for the entire region. The elements region-time-to-live or region-idle-time are deprecated.

API: See APIs for entry-time-to-live, entry-idle-time, region-time-to-live, region-idle-time

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to 0, (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 cluster. 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 cluster. 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

<custom-expiry>

Specifies the custom class that implements org.apache.geode.cache.CustomExpiry. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements.

API: org.apache.geode.cache.RegionFactory.setCustomEntryIdleTimeout, setCustomEntryTimeToLive

Example:

```
<region-attributes>
  <entry-time-to-live>
    <expiration-attributes
      timeout="60"
      action="local-destroy"/>
  </entry-time-to-live>
  <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.

---

entry-idle-time

<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.

> To ensure reliable read behavior across the partitioned region, use entry-time-to-live for entry expiration instead of this setting.

API: org.apache.geode.cache.RegionFactory.setEntryIdleTimeout

Example:

```
<region-attributes
  statistics-enabled="true">
  <entry-idle-time>
    <expiration-attributes
      timeout="60"
      action="local-invalidate"/>
  </entry-idle-time>
</region-attributes>
```

---

expiration-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 cluster. Within the region-time-to-live or region-idle-time element, this element specifies the expiration rules for the entire region. The elements region-time-to-live or region-idle-time are deprecated.

API: See APIs for entry-time-to-live, entry-idle-time, region-time-to-live, region-idle-time

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to 0 (zero), which means no expiration.	0

Attribute	Description	Default
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 cluster. `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 cluster. `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-idle-time>
    <expiration-attributes
      timeout="60"
      action="local-destroy"/>
  </entry-idle-time>
</region-attributes>
```

custom-expiry

`<custom-expiry>`

Specifies the custom class that implements org.apache.geode.cache.CustomExpiry. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: org.apache.geode.cache.RegionFactory.setCustomEntryIdleTimeout, setCustomEntryTimeToLive

Example:

```
<region-attributes>
  <entry-idle-time>
    <expiration-attributes
      timeout="60"
      action="local-destroy"/>
  </entry-idle-time>
  <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.

---

partition-attributes

<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.

> With the exception of local-max-memory, all members defining a partitioned region must use the same partition attribute settings.

API: org.apache.geode.cache.RegionFactory.setPartitionAttributes

<partition-attributes> 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, Tanzu GemFire issues a warning, but operation continues. Besides setting the maximum memory to use for the member, this setting also tells Tanzu 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, Tanzu 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 deactivates local data caching.	90% (of local heap)
recovery-delay	Applies when redundant-copies is greater than 0. The number of milliseconds to wait after a member crashes before reestablishing redundancy for the region. A setting of -1 deactivates 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 to 3. If you specify 1, this partitioned region maintains the original and one backup, for a total of two copies. A value of 0 deactivates redundancy.	0
startup-recovery-delay	Applies when redundant-copies is greater than 0. 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 deactivates automatic recovery of redundancy after new members join.	0

Attribute	Description	Default
total-max-memory	Maximum combined megabytes of memory to be used by all processes hosting this region for all copies, primary and redundant.	Integer.MAX_VALUE
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 equals 2 or 3 .	113

Example:

```
<region-attributes>
  <partition-attributes
    redundant-copies="1"
    total-num-buckets= "613"/>
</region-attributes>
```

partition-resolver

<partition-resolver>

Describes a custom PartitionResolver for a region.

API: org.apache.geode.cache.PartitionAttributesFactory.setPartitionResolver

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See <class-name> and <parameter>.

<partition-resolver> 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

<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>.

API: org.apache.geode.cache.PartitionAttributesFactory.PartitionListener

Example:

```
<partition-attributes redundant-copies="1">
  <partition-listener>
    <class-name>com.myCompany.ColocatingPartitionListener</class-name>
    <parameter name="viewRegions">
      <string>/customer/ViewA,/customer/ViewB</string>
    </parameter>
  </partition-listener>
</partition-attributes>
```

---

fixed-partition-attributes

<fixed-partition-attributes>

Fixed custom partitioning is deprecated and will be removed in upcoming releases. Describes a partition in a Fixed Partitioned Region.

API: org.apache.geode.cache.PartitionAttributesFactory.addFixedPartitionAttributes

<fixed-partition-attributes> 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-resolver>
      </partition-attributes>
    </region-attributes>
  </region>
</cache>
```

---

membership-attributes

<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: org.apache.geode.cache.RegionFactory.setMembershipAttributes

<membership-attributes> 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. 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
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: 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 perform a new getInitialImage operation to repopulate the region.	reinitialize

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

<required-role>

Specifies a role that is required for reliable access to the region.

API: org.apache.geode.cache.MembershipAttributes

<required-role> 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

<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.

> The interest policy determines which events are delivered, but the data-policy determines how the events are applied to the cache.

API: org.apache.geode.cache.RegionFactory.setSubscriptionAttributes

<subscription-attributes> Attributes

Attribute	Description	Default
interest-policy	Set one of the following interest-policy options: **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. **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

<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: org.apache.geode.cache.RegionFactory.setCacheLoader

Example:

```
<region-attributes>
  <cache-loader>
    <class-name>quickstart.SimpleCacheLoader</class-name>
  </cache-loader>
</region-attributes>
```

---

cache-writer

<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 cancel 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: org.apache.geode.cache.RegionFactory.setCacheWriter

Example:

```
<region-attributes>
  <cache-writer>
    <class-name>quickstart.SimpleCacheWriter</class-name>
  </cache-writer>
</region-attributes>
```

---

cache-listener

<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. Tanzu 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>.

API: org.apache.geode.cache.RegionFactory.addCacheListener

Example:

```
<region-attributes>
  <cache-listener>
    <class-name>
      quickstart.SimpleCacheListener
    </class-name>
  </cache-listener>
</region-attributes>
```

---

compressor

<compressor> A compressor registers a custom class that extends Compressor to support compression on a region.

Example:

```
 ...
<compressor>
  <class-name>
    <parameter>
      <string>
      <declarable>
  ...
</compressor>
...
```

---

eviction-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 stored region attributes.

Default: Uses the lru-entry-count algorithm.

API: org.apache.geode.cache.RegionFactory.setEvictionAttributes

Example:

```
<region-attributes>
  <eviction-attributes>
    <lru-entry-count
      maximum="1000"
      action="overflow-to-disk">
  </eviction-attributes>
</region-attributes>
```

---

lru-entry-count

<lru-entry-count>

Using the maximum attribute, specifies maximum region capacity based on entry count.

<lru-entry-count> Attributes

Attribute	Description	Default
action	Set one of the following eviction actions: local-destroy: Removes the entry from the local cache, but does not distribute the removal operation to remote members. This action can be applied to an entry in a partitioned region, but is not recommended if redundancy is enabled (redundant-copies greater than 0) because doing so introduces inconsistencies between the redundant buckets. When applied to an entry in a replicated region, Tanzu GemFire silently changes the region type to preloaded to accommodate the local modification. overflow-to-disk: The entry's value is overflowed to disk and set to null in memory. The entry's key is retained in the cache.	local-destroy
maximum	The maximum number of entries allowed in a region.	

lru-heap-percentage

<lru-heap-percentage>

Runs evictions when the Tanzu GemFire resource manager says to. The manager orders evictions when the total cache size is over the heap or off-heap percentage limit specified in the resource manager configuration.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See <class-name> and <parameter>.

<lru-heap-percentage> Attributes

Attribute	Description	Default
action	Set one of the following eviction actions: local-destroy: Removes the entry from the local cache, but does not distribute the removal operation to remote members. This action can be applied to an entry in a partitioned region, but is not recommended if redundancy is enabled (redundant-copies greater than 0) because doing so introduces inconsistencies between the redundant buckets. When applied to an entry in a replicated region, Tanzu GemFire silently changes the region type to preloaded to accommodate the local modification. overflow-to-disk: The entry's value is overflowed to disk and set to null in memory. The entry's key is retained in the cache.	local-destroy

lru-memory-size

<lru-memory-size>

Using the maximum attribute, specifies maximum region capacity based on the amount of memory used, in megabytes.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See <class-name> and <parameter>.

<lru-memory-size> Attributes

Attribute	Description	Default
action	Set one of the following eviction actions: local-destroy: Removes the entry from the local cache, but does not distribute the removal operation to remote members. This action can be applied to an entry in a partitioned region, but is not recommended if redundancy is enabled (redundant-copies greater than 0) because doing so introduces inconsistencies between the redundant buckets. When applied to an entry in a replicated region, Tanzu GemFire silently changes the region type to preloaded to accommodate the local modification. overflow-to-disk: The entry's value is overflowed to disk and set to null in memory. The entry's key is retained in the cache.	local-destroy
maximum	The maximum amount of memory used in the region, in megabytes.	

<jndi-bindings

<jndi-bindings>

Specifies the binding for a data-source used in transaction management. See Configuring Database Connections Using JNDI.

Example:

```
<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>CreateDatabase</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value>create</config-property-value>
    </config-property>
    ...
  </jndi-binding>
</jndi-bindings>
```

---

jndi-binding

<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. Tanzu GemFire uses the attributes of the <jndi-binding> element for configuration. Use the <config-property> element to configure properties for the datasource.

We recommend that you set the username and password with the user-name and password jndi-binding attributes rather than using the <config-property> element.

<jndi-binding> 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 PooledDataSource type.	

Attribute	Description	Default
connection-url	URL for connecting to the datasource. Note: If you are connecting to a JCA data source driver that implements XA transactions (where the jndi-binding type is XAPooledDataSource), do not use this attribute. Instead, define configuration properties for your database. See <config-property> for an example.	
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 PooledConnection (an XAConnection or a non-XAConnection).	10
jdbc-driver-class	Java class used for the SimpleDataSource type.	
jndi-name	The jndi-name attribute 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, Tanzu GemFire logs a warning.	
login-timeout-seconds	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 PoolException is thrown. If a connection is available the thread returns immediately. 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. If a connection is not available, the thread blocks for the specified time while waiting for an available connection.	30
managed-conn-factory-class	If the Resource Adapter is of type ManagedDataSource, this class becomes the source of the PooledConnection. (This class interface complies with the J2CA Java 2 Connector Architecture.)	
max-pool-size	The maximum size of the PooledConnection.	30

Attribute	Description	Default
password	Password to access the datasource.	
transaction-type	When the type attribute is set to ManagedDataSource, specifies the type of transaction. Set one of the following transaction-types: XA Transaction: Select this option when you want to use aManagedConnection interface with a Java Transaction Manager to define transaction boundaries. This option allows a ManagedDataSource to participate in a transaction with a Tanzu GemFire cache. No Transaction: No transactional behavior is used. Local Transaction: Select this option when using a ManagedDataSource that is not managed by the Java Transaction manager.	none
type	Set one of the following types: XAPooledDataSource: Pooled SQL connections. For this type, you must also set the xa-datasource-class attribute. ManagedDataSource: JNDI binding type for the J2EE Connector Architecture (JCA). ManagedConnectionFactory. For information about the ManagedConnection interface, For more information, see the Oracle ManagedConnection documentation. PooledDataSource: Pooled SQL connections. For this type, you must also set the conn-pooled-datasource-class 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 jdbc-driver-class attribute.	none

Attribute	Description	Default
user-name	User name to access to the datasource.	
xa-datasource-class	Java class used for the XAPooledDataSource type.	

config-property

<config-property>

A configuration property of the datasource. Use the sub-elements to identify the name, datatype, and value of the property.

Default:

Example:

```
<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>
```

Configuration properties vary depending on the database vendor. See Configuring Database Connections Using JNDI for examples of different configuration property configurations.

config-property-name

<config-property-name>

The name of this datasource property.

config-property-type

<config-property-type>

The data type of this datasource property.

config-property-value

<config-property-value>

The value of this datasource property.

region

<region>

Defines a region in the cache. For more details about configuring regions, see <region-attributes>. You can specify zero or more subregions within a region. For restrictions on creating subregions, see Create and Access Data Subregions. For example, you cannot create a partitioned subregion.

Default:

API: org.apache.geode.cache.RegionFactory or org.apache.geode.cache.ClientRegionFactory

<region> Attributes

Attribute	Description	Default
name	Specify the name for the region. See Region Management for details.	
refid	Used to apply predefined attributes to the region being defined. If the nested region-attributes element has its own refid, the refid on the region is ignored. The refid region attribute can be set to the name of a RegionShortcut or a ClientRegionShortcut. For more information, see Region Shortcuts and Custom Named Region Attributes and Storing and Retrieving Region Shortcuts and Custom Named Region Attributes.	

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"/>
```

For a complete listing of region attributes, see <region-attributes>.

index

<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 for more information about indexes.

Default:

API: org.apache.geode.cache.query.QueryService.createIndex, createKeyIndex, createHashIndex

<index> Attributes

Attribute	Description	Default
name	Required. Name of the index.	
from-clause	Specifies the collections 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.	

Attribute	Description	Default
key-index	Boolean that determines if 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>
```

---

lucene:index

<lucene:index>

Describes a Lucene index to be created on a region. The lucene namespace and the scoping operator (:) must be specified, as the Tanzu GemFire cache namespace also defines an index element (for OQL indexes).

API: org.apache.geode.cache.lucene package

<lucene:index> Attributes

Attribute	Description	Default
name	Required. Name of the Lucene index.	

Example:

```
<cache
  xmlns="http://geode.apache.org/schema/cache"
  xmlns:lucene="http://geode.apache.org/schema/lucene"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://geode.apache.org/schema/cache
    http://geode.apache.org/schema/cache/cache-1.0.xsd
    http://geode.apache.org/schema/lucene
    http://geode.apache.org/schema/lucene/lucene-1.0.xsd"
  version="1.0">

  <region name="regionA" refid="PARTITION">
    <lucene:index name="myIndex">
      <lucene:field name="x" />
      <lucene:field name="y" />
    </lucene:index>
  </region>
</cache>
```

---

lucene:field

<lucene:field>

Describes a field to be included in a Lucene index. Including the lucene namespace and the scoping operator (:) clarifies, but is not required.

API: org.apache.geode.cache.lucene package

<lucene:field> Attributes

Attribute	Description	Default
name	Required. A string that defines the name of the field. If a single field is defined by the value "__REGION_VALUE_FIELD", then the entire value is used as a single field.	
analyzer	A string that provides the path to the analyzer to use for this field. A value of "null" uses the default analyzer.	"null"

Example:

```
<region name="dataregion" refid="PARTITION_REDUNDANT">
  <lucene:index name="full_value_index">
    <lucene:field name="__REGION_VALUE_FIELD"/>
  </lucene:index>
</region>
```

---

entry

<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: org.apache.geode.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

<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

<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.

API: Declarable

Example:

```
<parameter name="cacheserver">
  <declarable>
    <class-name>org.apache.geode.addon.cache.CacheServerInitializer</class-name>
    <parameter name="system.property.prefix">
      <string>cacheserver</string>
    </parameter>
  </declarable>
</parameter>
```

---

value

<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

<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.

API: Declarable

Example:

```
<parameter name="cacheserver">
  <declarable>
    <class-name>org.apache.geode.addon.cache.CacheServerInitializer</class-name>
    <parameter name="system.property.prefix">
      <string>cacheserver</string>
    </parameter>
  </declarable>
</parameter>
```

---

region

<region>

When nested within a <region> element, defines a subregion. For restrictions on creating subregions, see Create and Access Data Subregions. For example, you cannot create a partitioned subregion.

See <region>

---

function-service

<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

<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>.

Default:

API: org.apache.geode.cache.execute.FunctionService

Example:

```
<function>
 <class-name>
  com.myCompany.tradeService.cache.func.TradeCalc
 </class-name>
</function>
```

---

resource-manager

<resource-manager>

A memory monitor that tracks cache size as a percentage of total heap or off-heap memory and controls size by restricting access to the cache and prompting eviction of old entries from the cache. For tenured heap, used in conjunction with settings for JVM memory and Java garbage collection. For off-heap memory, used with the off-heap memory manager.

API: org.apache.geode.cache.control.ResourceManager

<resource-manager> 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. 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. For details, see setCriticalHeapPercentage().	0

Attribute	Description	Default
eviction-heap-percentage	Set the percentage of heap at or above which the eviction should begin on regions configured for HeapLRU eviction. Changing this value may cause eviction to begin immediately. • The default is 0, if no region is configured with heap eviction • If critical-heap-percentage is set to a non-zero value, 5% less than that value. • 80%, if critical-heap-percentage is not configured.	0
critical-off-heap-percentage	Percentage of off-heap memory 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-off-heap-percentage	Set the percentage of off-heap memory at or above which the eviction should begin on Regions configured for HeapLRU eviction. • The default is 0, if no region is configured with heap eviction • If critical-off-heap-percentage is set to a non-zero value, 5% less than that value. • 80%, if critical-off-heap-percentage is not configured.	0

Example:

```
<cache>
  ...
  <resource-manager
    critical-heap-percentage="99.9"
    eviction-heap-percentage="85"/>
  ...
</cache>
```

serialization-registration

<serialization-registration>

Set of serializer or instantiator tags to register customer DataSerializer extensions or DataSerializable implementations respectively.

Example:

```
<serialization-registration>
  <instantiator id="30">
    <class-name>com.package.MyClass</class-name>
  </instantiator>
</serialization-registration>
```

serializer

<serializer>

Allows you to configure the DataSerializer for this Tanzu GemFire member. It registers a custom class which extends DataSerializer to support custom serialization of non-modifiable object types inside Tanzu 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 org.apache.geode.DataSerializer.register API. Use the org.apache.geode.Instantiator API to register a DataSerializable implementation.

---

instantiator

<instantiator>

An Instantiator registers a custom class which implements the DataSerializable interface to support custom object serialization inside Tanzu 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 org.apache.geode.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>.

<instantiator> Attributes

Attribute	Description	Default
id	Required. ID that the Instantiator should associate with the DataSerializable type.	

---

backup

<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/exampleuser/gfSystemInfo/myCustomerConfig.doc</backup>
```

---

initializer

<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>.

Default:

API: Declarable

Example:

```
<initializer>
  <class-name>MyInitializer</class-name>
  <parameter name="members">
    <string>2</string>
  </parameter>
</initializer>
```

---

declarable

<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.

API: Declarable

Example:

```
<parameter name="cacheserver">
  <declarable>
    <class-name>org.apache.geode.addon.cache.CacheServerInitializer</class-name>
    <parameter name="system.property.prefix">
      <string>cacheserver</string>
    </parameter>
  </declarable>
</parameter>
```

---

class-name and 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> sub-element or by specifying a Java class with the <declarable> sub-element.

Example:

The following transaction writer configuration example specifies a Java class named com.company.data.MyTransactionWritet. 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

<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.

API: Declarable

Example:

```
<parameter name="cacheserver">
  <declarable>
    <class-name>org.apache.geode.addon.cache.CacheServerInitializer</class-name>
    <parameter name="system.property.prefix">
      <string>cacheserver</string>
    </parameter>
  </declarable>
</parameter>
```

---

string

<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>
```

Element Hierarchy

This topic shows the hierarchy of <client-cache> element sub-elements that you use to configure VMware Tanzu GemFire caches and clients.

For details, see <client-cache> Element Reference.

```
<client-cache>
  <cache-transaction-manager>
    <transaction-listener>
    <transaction-writer>
  <pool>
    <locator>
    <server>
    <socket-factory>
  <disk-store>
    <disk-dirs>
      <disk-dir>
  <pdx>
    <pdx-serializer>
      <class-name>
      <parameter>
        <string>
        <declarable>
  <region-attributes>
    <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>
    <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>
    <jndi-binding>
      <config-property>
        <config-property-name>
        <config-property-type>
        <config-property-value>
  <region>
    <region-attributes>
    <index>
      <entry>
      <key>
        <string>
        <declarable>
      <value>
        <string>
        <declarable>
    <region>
  <function-service>
    <function>
      <class-name>
      <parameter>
        <string>
        <declarable>
  <resource-manager>
  <serialization-registration>
    <serializer>
      <class-name>
    <instantiator>
      <class-name>
  <initializer>
    <class-name>
    <parameter>
      <string>
      <declarable>
</client-cache>
```

Element Reference

This topic documents all of the cache.xml elements that you use to configureVMware Tanzu GemFire clients. All elements are sub-elements of the <client-cache> element.

For Tanzu GemFire server configuration, see <cache> Element Reference.

API: org.apache.geode.cache.client.ClientCacheFactory and PoolFactory interfaces.

client-cache Attributes

client-cache Attributes

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

<cache-transaction-manager>

Specifies a transaction listener.

API: CacheTransactionManager

Example:

```
<client-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> ...
</client-cache>
```

transaction-listener

<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>.

transaction-writer

<transaction-writer>

When you commit a transaction, a TransactionWriter can perform additional tasks, including cancelling the transaction.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See <class-name> and <parameter>.

pool

<pool>

Use for client caches. Defines a client's server pool used to communicate with servers running in a different cluster.

API: org.apache.geode.cache.client.PoolFactory

<pool> Attributes

Attribute	Description	Default
free-connection-timeout	Amount of time, in milliseconds, a thread will wait to get a pool connection before timing out with an exception. This timeout keeps threads from waiting indefinitely when a pool's max-connections has been reached and all connections in the pool are in use by other threads. Takes effect only when pr-single-hop-enabled is false. When pr-single-hop-enabled is true, use server-connection-timeout.	10000 (10 seconds)
idle-timeout	Maximum time, in milliseconds, a pool connection can stay open without being used when there are more than min-connections in the pool. Pings over the connection do not count as connection use. If set to -1, there is no idle timeout. If load-conditioning-interval is set to a nonzero value, then idle-timeout cannot exceed that value. In this case, if idle-timeout is set to -1 or to a value greater than the load-conditioning-interval, it is reset to the same value as load-conditioning-interval.	120000 (2 minutes)
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. VMware recommends that you set load-polling-interval higher than the server's load-poll-interval and the client's ping-interval.	300000 (5 minutes)

Attribute	Description	Default
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 free-connection-timeout is reached. If set to -1, there is no maximum. The setting must indicate a cap greater than min-connections. Takes effect only when pr-single-hop-enabled is false. When pr-single-hop-enabled is true, use max-connections-per-server.	-1
max-connections-per-server	Maximum number of connections pooled to a single server. If the maximum connections are in use, an operation requiring a client-to-server connection to that server blocks until a connection becomes available or server-connection-timeout is reached. If set to -1, there is no maximum. The setting must indicate a cap greater than min-connections-per-server. Takes effect only when pr-single-hop-enabled is true. When pr-single-hop-enabled is false, use max-connections.	-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 max-connections setting. The setting must be an integer greater than or equal to 0.	1
min-connections-per-server	Minimum number of connections for each cache server in the cluster. The system tries to keep available at least this many connections to each server at all times. When a new pool is created, it immediately begins creating connections to each server, up to this minimum. 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 max-connections-per-server 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 RegionService instance. If set to false, the client either uses no authorization or only 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. Note: 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. Note: Set this lower than the server's maximum-time-between-pings.	10000

Attribute	Description	Default
pr-single-hop-enabled	Optimizes access to partitioned region data in the servers. When true, the pool configuration is also affected by the attributes max-connections-per-server, min-connections-per-server, and server-connection-timeout. When false, the pool configuration is also affected by the attributes max-connections, min-connections, and free-connection-timeout. For details about how these settings interact, see Configuring a Client/Server System.	true
read-timeout	Maximum time, in milliseconds, for the client to wait for a response from a server.	10000 (10 seconds)
retry-attempts	Number of times to retry a client request before giving up. If one server fails, the pool moves to the next, and continues to move to the next until it is successful or it meets this limit. If the number of available servers is lower 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-connection-timeout	Time, in milliseconds, that a thread will wait to get a pool connection to a specific server, before timing out with an exception. This timeout keeps threads from waiting indefinitely when the pool's max-connections-per-server has been reached and all connections in a pool are in use by other threads.	0
server-group	Logical named server group to use from the pool. A null value uses the global server group to which all servers belong. Note: This is only used when the locator list is defined.	null
socket-buffer-size	Size for socket buffers from the client to the server.	32768
socket-connect-timeout	Time, in milliseconds, to wait before timing out on the client socket connect to a server or locator. A value of 0 (zero) indicates an infinite timeout. If set to 0, the connection blocks until established or an error occurs.	59000
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. Note: Used only when subscription-redundancy is not 0 (zero).	100
subscription-enabled	Boolean that indicates 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

Attribute	Description	Default
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, no servers are used. If set to -1, all available servers are used.	0
subscription-timeout-multiplier	Number of missing server pings that trigger timeout of a subscription feed. A value of 0 (zero) deactivates timeouts. A value of 1 or higher sets the server connection to time out after the specified number of ping intervals have elapsed. VMware recommends that you do not set this to a value of 1.	0

Example:

```
<pool
  name="publisher"
  subscription-enabled="true">
  <locator
    host="myLocatorAddress1"
    port="12345"/>
   <locator
     host="myLocatorAddress2"
     port="45678"/>
</pool>
```

locator

<locator>

Addresses and ports of the locators to connect to. You can define multiple locators for the pool.

> Provide a locator list or server list, but not both.

<locator> Attributes

Attribute	Description	Default
host	Hostname of the locator	
port	Port number of the locator	

Example:

```
<pool ...>
  <locator
    host="myLocatorHost"
    port="12345"/>
```

server

<server>

Addresses and ports of the servers to connect to.

> Provide a server list or locator list, but not both.

Default:

<server> Attributes

Attribute	Description	Default
host	Hostname of the server	
port	Port number of the server	

Example:

```
<pool ...>
  <server
    host="myServerHost"
    port="123456"/>
</pool>
```

---

socket-factory

<socket-factory>

Defines a factory to create socket connections to locators and servers. A typical use of this element is to redirect connections to an ingress gateway such as Istio or HAProxy in a cluster where the TLS (SSL) Server Name Indication (SNI) field is set to indicate the actual locator or server the client is trying to reach. This allows you to expose only the gateway hostname:port without the client needing to be able to resolve the names of the locator and server machines.

> This setting may be used with either a Server list or a Locator list. It will be used to form connections to either.

Default:

API: org.apache.geode.cache.client.proxy.ProxySocketFactories

Example:

```
<pool ...>
  <socket-factory>
    <class-name>org.apache.geode.cache.client.proxy.SniProxySocketFactory</class-name>
    <parameter name="hostname">
      <string>my-gateway-address</string>
    </parameter>
    <parameter name="port">
      <string>12345</string>
    </parameter>
  </socket-factory>
</pool>
```

---

disk-store

<disk-store>

Defines a pool of one or more disk stores, which can be used by regions, and client subscription queues.

Default: The cache default disk store, named “DEFAULT”, is used when disk is used but no disk store is named.

API: org.apache.geode.cache.DiskStore

<disk-store> 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 operation log (oplog) will become able to be compacted. Until this threshold is reached, 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 operation log (oplog) file.	
time-interval	Time, in 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.	
disk-usage-warning-percentage	Disk usage above this threshold generates a warning message. For example, if the threshold is set to 90%, then on a 1 TB drive falling under 100 GB of free disk space generates the warning. Set to 0 (zero) to deactivate.	90
disk-usage-critical-percentage	Disk usage above this threshold generates an error message and shuts down the member's cache. For example, if the threshold is set to 99%, then falling under 10 GB of free disk space on a 1 TB drive generates the error and shuts down the cache. Set to 0 (zero) to deactivate.	99

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

<disk-dirs>

An element of a disk store that defines a set of <disk-dir> elements.

---

disk-dir

<disk-dir>

Specifies a region or disk store's disk directory.

<disk-dir> 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

<pdx>

Specifies the configuration for the Portable Data eXchange (PDX) method of serialization.

API: org.apache.geode.cache.CacheFactory.setPdxReadSerialized, setPdxDiskStore, setPdxPersistent, setPdxIgnoreUnreadFields and org.apache.geode.cache.ClientCacheFactory.setPdxReadSerialized, setPdxDiskStore, setPdxPersistent, setPdxIgnoreUnreadFields

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. 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:

```
<client-cache>
 <pdx persistent="true" disk-store-name="myDiskStore">
  <pdx-serializer>
   <class-name>
    org.apache.geode.pdx.ReflectionBasedAutoSerializer
   </class-name>
   <parameter name="classes">
    <string>com.company.domain.DomainObject</string>>
   </parameter>
  </pdx-serializer>
 </pdx>
 ...
</client-cache>
```

---

pdx-serializer

<pdx-serializer>

Allows you to configure the PdxSerializer for this Tanzu GemFire member.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements. See <class-name> and <parameter>.

Default:

API: org.apache.geode.cache.CacheFactory.setPdxSerializer

Example:

```
<client-cache>
 <pdx>
  <pdx-serializer>
   <class-name>com.company.ExamplePdxSerializer</class-name>
  </pdx-serializer>
 </pdx>
 ...
</client-cache>
```

---

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: org.apache.geode.cache.RegionFactory or org.apache.geode.cache.ClientRegionFactory

<region-attributes> Attributes

Attribute	Description	Default

concurrency-level	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 Tanzu 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.
	
	Note: Before you modify this, read the concurrency level description, then see the Java API documentation for java.util.ConcurrentHashMap.
	
	API: setConcurrencyLevel
	
	Example:
	
	`<region-attributes`
	`  concurrency-level="10">`
	`</region-attributes>`
		16 (threads)

Attribute	Description	Default
data-policy	Specifies how the local cache handles data for a region. This setting controls behavior such as local data storage and region initialization. Note: 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. You can specify the following data policies: 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 subscription-attributes interest-policy to all. normal: Data used locally (accessed with get, stored with put, etc.) is stored in the local cache. This policy allows the contents in the cache to differ from other caches. partition: Data is partitioned across local and remote caches using the automatic data distribution behavior of partitioned regions. Additional configuration is done in the partition-attributes. 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 scope, behaving the same as for normal. persistent-partition: Behaves the same as partition and also persists data to disk. persistent-replicate: Behaves the same as replicate and also persists data to disk. preloaded: Initializes like a replicated region. After initialization, behaves like a normal region. API: setDataPolicy	normal

Attribute	Description	Default

	Example:
	
	``<region-attributes
	  data-policy="replicate">
	</region-attributes>```
	
	This is similar to using a region shortcut with refid, however when you use the REPLICATE region shortcut, it automatically sets the region's scope to distributed-ack.
	
	``<region-attributes
	  refid="REPLICATE">
	</region-attributes>```
	
	If you use data-policy, you must set the scope explicitly.	

enable-async-conflation	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 deactivates 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. Note: To use this attribute, asynchronous queues must be enabled for slow consumers, specified with the async* gemfire properties. API: setEnableAsyncConflation Example: ```<region-attributes enable-async-conflation="false"> </region-attributes>``` Determines whether the gateway is enabled for the region. When set to true, events in the region are sent to the defined gateway hubs. Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the gateway-sender-id attribute of the <region-attributes> element.	true
enable-gateway		false

Attribute	Description	Default

enable-subscription-conflation	Boolean for server regions that specifies whether the server can conflate its messages to the client. A value of true enables conflation.
	
	Note: The client can override this setting with the conflate-events property in its gemfire.properties.
	
	API: setEnableSubscriptionConflation
	
	Example:
	
	`<region-attributes enable-subscription-conflation="true">`
	`</region-attributes>`
		false

gateway-sender-ids	Specifies one or more gateway sender IDs to use for distributing region events to remote Tanzu GemFire sites. Specify multiple IDs as a comma-separated list.
	
	API: addGatewaySenderId
	
	Example:
	
	`<region-attributes gateway-sender-ids="nwsender,swsender">`
	`</region-attributes>`
		not set

async-event-queue-ids	Specifies one or more asynchronous event queues to use for distributing region events an AsyncEventListener implementation (for example, for write-behind cache event handling). Specify multiple IDs as a comma-separated list.
	
	API: addAsyncEventQueueId
	
	Example:
	
	`<region-attributes async-event-queue-ids="customerqueue,ordersqueue">`
	`</region-attributes>`
		not set

hub-id	If the enable-gateway attribute is set to true, a comma-separated list of gateway hub IDs that receive events from the region. Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the gateway-sender-id attribute of the <region-attributes> element.	null

Attribute	Description	Default

id	Stores the region attribute settings in the cache with this identifier. Once stored, the attributes can be retrieved using the region attribute refid.
	
	API: setId
	
	Example:
	
	`<region-attributes id="persistent-replicated">`
	`</region-attributes>`
		not set

ignore-jta	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.
	
	API: setIgnoreJTA
	
	Example:
	
	`<region-attributes ignore-jta="true">`
	`</region-attributes>`
		false

index-update-type	Specifies whether region indexes are maintained synchronously with region modifications, or asynchronously in a background thread. In the cache.xml file, this is set as a value, asynchronous or synchronous, assigned to the index-update-type region attribute. Set this through the API by passing a boolean to the setIndexMaintenanceSynchronous method.
	
	API: setIndexMaintenanceSynchronous
	
	Example:
	
	`<region-attributes index-update-type="asynchronous">`
	`</region-attributes>`
		synchronous updates

initial-capacity	Together with the load-factor region attribute, sets the initial parameters on the underlying java.util.ConcurrentHashMap used for storing region entries.
	
	API: setInitialCapacity
	
	Example:
	
	`<region-attributes initial-capacity="20">`
	`</region-attributes>`
		16

Attribute	Description	Default

is-lock-grantor	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 is-lock-grantor set to true. This attribute is only relevant for regions with global scope, as only they allow locking. It affects implicit and explicit locking.
	
	API: setLockGrantor
	
	Example:
	
	`<region-attributes`
	`  is-lock-grantor="true">`
	`</region-attributes>`
		false

load-factor	Together with the initial-capacity region attribute, sets the initial parameters on the underlying java.util.ConcurrentHashMap used for storing region entries. This must be a floating point number from 0 and 1, inclusive.
	
	Note: Before you set this attribute, VMware recommends that you read the discussion of initial capacity and load factor, then see the Java API documentation for java.util.ConcurrentHashMap.
	
	API: setLoadFactor
	
	Example:
	
	`<region-attributes`
	`  load-factor="0.85">`
	`</region-attributes>`
		.75

mirror-type	Deprecated	

Attribute	Description	Default

pool-name	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.
	
	API: setPoolName
	
	Examples:
	
	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:
	
	`<client-cache>`
	`  <pool name="DatabasePool"`
	`  subscription-enabled="true">`
	`  ...`
	`  </pool>`
	`  <pool >`
	`  name="OtherPool"`
	`  subscription-enabled="true">`
	`  ...`
	`  </pool>`
	`  <region ...>`
	`  <region-attributes`
	`  pool-name="DatabasePool">`
	`  </region-attributes>`
	`  ...`
	
	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:
	
	`<client-cache>`
	`  <pool`
	`  name="publisher"`
	`  subscription-enabled="true">`
	`  ...`
	`  </pool>`
	`  <region`
	`  name="myRegion"`
	`  refid="CACHING_PROXY">`
	`  </region>`
	`</client-cache>`
		not set

Attribute	Description	Default

disk-store-name	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 disk-synchronous region attribute specifies whether writes are done synchronously or asynchronously.
	API: setDiskStoreName
	Example:
	`<region-attributes`
	`  disk-store-name="myStoreA" >`
	`</region-attributes>`
		null

disk-synchronous	For regions that write to disk, boolean that specifies whether disk writes are done synchronously for the region.
	API: setDiskSynchronous
	Example:
	`<region-attributes`
	`  disk-store-name="myStoreA"`
	`  disk-synchronous="true">`
	`</region-attributes>`
		true

refid	Retrieves region shortcuts and user-defined named region attributes for attributes initialization
	API: setRefId
	Example:
	`<region-attributes`
	`  refid="persistent-replicated">`
	`  <!-- Override any stored`
	`  attribute settings that you`
	`  need to ... -->`
	`</region-attributes>`
		not set

Attribute	Description	Default

scope	Determines how updates to region entries are distributed to the other caches in the cluster 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.
	
	Note: You can configure the most common of these options with Tanzu GemFire region shortcuts in RegionShortcut and ClientRegionShortcut.
	
	Note: Server regions that are not partitioned must be replicated with distributed-ack or global scope. The region shortcuts that specify REPLICATE have distributed-ack scope.
	
	Set one of the following scope values:
	
	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 cluster.
	
	global: Events are distributed to remote caches with global locking to ensure distributed cache consistency.
	
	API: setScope
	
	Example:
	
	`<region-attributes scope="distributed-ack">`
	`</region-attributes>`
		distributed-no-ack

Attribute	Description	Default

statistics-enabled	Boolean that specifies whether to gather statistics on the region. Must be true to use expiration on the region. Tanzu GemFire provides a standard set of statistics for cached regions and region entries, which give you information for fine-tuning your cluster. Unlike other Tanzu GemFire statistics, statistics for local and distributed regions are not archived and cannot be charted. They are kept in instances of org.apache.geode.cache.CacheStatistics and made available through the region and its entries through the Region.getStatistics and Region.Entry.getStatistics methods.
	
	API: setStatisticsEnabled
	
	Example:
	
	`<region-attributes statistics-enabled="true">`
	`</region-attributes>`
		false

cloning-enabled	Boolean used to determine how fromDelta applies deltas to the local cache for delta propagation. When set to true, the updates are applied to a clone of the value and then the clone is saved to the cache. When set to false, the value is modified in place in the cache.
	
	API: setCloningEnabled
	
	Example:
	
	`<region-attributes cloning-enabled="true">`
	`</region-attributes>`
		false

concurrency-checks-enabled	Boolean used to determine whether members perform checks to provide consistent handling for concurrent or out-of-order updates to distributed regions. See Consistency for Region Updates.
	
	Note: Applications that use a client-cache may want to deactivate concurrency checking to see all events for a region. Tanzu 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 unsynchronized with the server cache.
	
	API: setConcurrencyChecksEnabled
	
	Example:
	
	`<region-attributes concurrency-checks-enabled="true">`
	`</region-attributes>`
		true

Attribute	Description	Default
off-heap	Boolean used to determine whether the region uses off-heap memory to store entry values, including values for region entries and queue entries. The region continues to use heap memory for everything else, such as entry keys and the ConcurrentHashMap. API: setOffHeap Example: <region-attributes off-heap="true"> </region-attributes>	false

---

key-constraint

<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 used for querying and indexing because it provides object type information to the query engine.

> Set the constraint in every cache where you create or update the region entries. For client/server installations, match constraints between client and server and between clusters. 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: org.apache.geode.cache.RegionFactory.setKeyConstraint

Example:

```
<region-attributes>
  <key-constraint>
    java.lang.String
  </key-constraint>
</region-attributes>
```

---

value-constraint

<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 is not used, the region's value can be of any class. This attribute, along with key-constraint, is used for querying and indexing because it provides object type information to the query engine.

> Set the constraint in every cache where you create or update the region entries. For client/server installations, match constraints between client and server and between clusters. 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: org.apache.geode.cache.RegionFactory.setValueConstraint

Example:

```
<region-attributes>
  <value-constraint>
    cacheRunner.Portfolio
  </value-constraint>
</region-attributes>
```

---

region-time-to-live

<region-time-to-live>

Deprecated. 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: org.apache.geode.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

<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 cluster. Within the region-time-to-live or region-idle-time element, this element specifies the expiration rules for the entire region. The elements region-time-to-live or region-idle-time are deprecated.

API: See APIs for entry-time-to-live, entry-idle-time, region-time-to-live, region-idle-time

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to 0 (zero), which indicates 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 cluster. invalidate: Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. Use this option when the region or the entry is no longer valid for any application in the cluster. 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

<custom-expiry>

Specifies the custom class that implements org.apache.geode.cache.CustomExpiry. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements.

API: org.apache.geode.cache.RegionFactory.setCustomEntryIdleTimeout, setCustomeEntryTimeToLive

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.

---

region-idle-time

<region-idle-time>

Deprecated. Expiration setting that specifies how long the region can remain in the cache without anyone accessing it.

Default: not set - no expiration of this type

API: org.apache.geode.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

<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 cluster. Within the region-time-to-live or region-idle-time element, this element specifies the expiration rules for the entire region. The elements region-time-to-live or region-idle-time are deprecated.

API: See APIs for entry-time-to-live, entry-idle-time, region-time-to-live, region-idle-time

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to 0 (zero), which indicates no expiration.	0

Attribute	Description	Default
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 cluster. `invalidate` Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. Use this option when the region or the entry is no longer valid for any application in the cluster. `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

`<custom-expiry>`

Specifies the custom class that implements `org.apache.geode.cache.CustomExpiry`. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: `org.apache.geode.cache.RegionFactory.setCustomEntryIdleTimeout`, `setCustomeEntryTimeToLive`

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.

---

entry-time-to-live

<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: org.apache.geode.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

<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 cluster. Within the region-time-to-live or region-idle-time element, this element specifies the expiration rules for the entire region. The elements region-time-to-live or region-idle-time are deprecated.

API: See APIs for entry-time-to-live, entry-idle-time, region-time-to-live, region-idle-time

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Time, in seconds, before a region or an entry expires. If timeout is not specified, it defaults to 0 (zero), which indicates no expiration.	0

Attribute	Description	Default
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 cluster. `invalidate` Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. Use this option when the region or the entry is no longer valid for any application in the cluster. `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

`<custom-expiry>`

Specifies the custom class that implements org.apache.geode.cache.CustomExpiry. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: org.apache.geode.cache.RegionFactory.setCustomEntryIdleTimeout, setCustomeEntryTimeToLive

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.

---

entry-idle-time

<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.

> To ensure reliable read behavior across the partitioned region, use entry-time-to-live for entry expiration instead of this setting.

API: org.apache.geode.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

<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 cluster. Within the region-time-to-live or region-idle-time element, this element specifies the expiration rules for the entire region. The elements region-time-to-live or region-idle-time are deprecated.

API: See APIs for entry-time-to-live, entry-idle-time, region-time-to-live, region-idle-time

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Time, in seconds, before a region or an entry expires. If timeout is not specified, it defaults to 0 (zero), which indicates expiration.	0

Attribute	Description	Default
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 cluster. invalidate: Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. Use this option when the region or the entry is no longer valid for any application in the cluster. 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

<custom-expiry>

Specifies the custom class that implements org.apache.geode.cache.CustomExpiry. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the <class-name> and <parameter> sub-elements.

API: org.apache.geode.cache.RegionFactory.setCustomEntryIdleTimeout, setCustomeEntryTimeToLive

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.

---

cache-loader

<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, to have a cache loader you must 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: org.apache.geode.cache.RegionFactory.setCacheLoader

Example:

```
<region-attributes>
  <cache-loader>
    <class-name>quickstart.SimpleCacheLoader</class-name>
  </cache-loader>
</region-attributes>
```

---

cache-writer

<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 cancel 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: org.apache.geode.cache.RegionFactory.setCacheWriter

Example:

```
<region-attributes>
  <cache-writer>
    <class-name>quickstart.SimpleCacheWriter</class-name>
  </cache-writer>
</region-attributes>
```

cache-listener

<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. Tanzu 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>.

API: org.apache.geode.cache.RegionFactory.addCacheListener

Example:

```
<region-attributes>
  <cache-listener>
    <class-name>
      quickstart.SimpleCacheListener
    </class-name>
  </cache-listener>
</region-attributes>
```

eviction-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 stored region attributes.

Default: Uses the lru-entry-count algorithm.

API: org.apache.geode.cache.RegionFactory.setEvictionAttributes

Example:

```
<region-attributes>
  <eviction-attributes>
    <lru-entry-count
      maximum="1000"
      action="overflow-to-disk"/>
  </eviction-attributes>
</region-attributes>
```

lru-entry-count

<lru-entry-count>

Using the maximum attribute, specifies maximum region capacity based on entry count.

<lru-entry-count> 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

<lru-heap-percentage>

Runs evictions when the Tanzu 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>.

<lru-heap-percentage> 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

<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>.

<lru-memory-size> 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 amount of memory used in the region, in megabytes.	

jndi-bindings

<jndi-bindings>

Specifies the binding for a data-source used in transaction management. See Configuring Database Connections Using JNDI.

Example:

```
<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>CreateDatabase</config-property-name>
   <config-property-type>java.lang.String</config-property-type>
   <config-property-value>create</config-property-value>
  </config-property>
  ...
 </jndi-binding>
</jndi-bindings>
```

jndi-binding

<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. Tanzu 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 <jndi-binding> attributes instead of using the <config-property> element.

<jndi-binding> Attributes

Attribute	Description	Default
blocking-timeout-seconds	Time, in 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 PooledDataSource type.	
connection-url	URL for connecting to the datasource. Note: If you are connecting to a JCA data source driver that implements XA transactions (where the jndi-binding type is XAPooledDataSource), do not use this attribute. Instead, define configuration properties for your database. See <config-property> for an example.	
idle-timeout-seconds	The maximum time, in 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 PooledConnection (an XAConnection or a non-XAConnection).	10
jdbc-driver-class	Java class used for the SimpleDataSource type.	
jndi-name	The jndi-name attribute 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, Tanzu GemFire logs a warning.	
login-timeout-seconds	The maximum time, in 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 PoolException is thrown If a connection is available the thread returns immediately. 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. If a connection is not available, the thread blocks for the specified time while waiting for an available connection.	30
managed-conn-factory-class	If the Resource Adapter is of type ManagedDataSource, this class becomes the source of the PooledConnection. (This class interface complies with the J2CA Java 2 Connector Architecture.)	
max-pool-size	The maximum size of the PooledConnection.	30
password	Password to access the datasource.	

Attribute	Description	Default
transaction-type	When the type attribute is set to ManagedDataSource, specifies the type of transaction. Set one of the following type of transaction: XATransaction: Select this option when you want to use aManagedConnection interface with a Java Transaction Manager to define transaction boundaries. This option allows a ManagedDataSource to participate in a transaction with a Tanzu GemFire cache. NoTransaction: No transactional behavior is used. LocalTransaction: Select this option when using a ManagedDataSource that is not managed by the Java Transaction manager.	none
type	Set one of the following types: XAPooledDataSource: Pooled SQL connections. For this type, you must also set the xa-datasource-class attribute. ManagedDataSource: JNDI binding type for the J2EE Connector Architecture (JCA). ManagedConnectionFactory. For information about the ManagedConnection interface, see Interface ManagedConnection in the Oracle Java documentation. PooledDataSource: Pooled SQL connections. For this type, you must also set the conn-pooled-datasource-class 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 jdbc-driver-class attribute.	none
user-name	User name to access to the datasource.	
xa-datasource-class	Java class used for the XAPooledDataSource type.	

config-property

<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>DatabaseName</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
  <config-property-value>newDB</config-property-value>
</config-property>
```

Configuration properties vary depending on the database vendor. See Configuring Database Connections Using JNDI for examples of different configuration property configurations.

---

config-property-name

<config-property-name>

The name of this datasource property.

---

config-property-type

<config-property-type>

The data type of this datasource property.

---

config-property-value

<config-property-value>

The value of this datasource property.

---

region

<region>

Defines a region in the cache. See <region-attributes> for more details on configuring regions. You can specify zero or more subregions within a region. See Create and Access Data Subregions for restrictions on creating subregions. For example, you cannot create a partitioned subregion.

Default:

API: org.apache.geode.cache.RegionFactory or org.apache.geode.cache.ClientRegionFactory

<region> Attributes

Attribute	Description	Default
name	Specify the name for the region. See Region Management 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 Region Shortcuts and Custom Named Region Attributes and Storing and Retrieving Region Shortcuts and Custom Named Region Attributes.	

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.

---

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: org.apache.geode.cache.RegionFactory or org.apache.geode.cache.ClientRegionFactory

<region-attributes> Attributes

Attribute	Description	Default
concurrency-level	Provides 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 Tanzu 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. Note: Before you modify this, VMware recommends that you read the concurrency level description, then see the Java API documentation for java.util.ConcurrentHashMap. API: setConcurrencyLevel Example: `<region-attributes concurrency-level="10">` `</region-attributes>`	16 (threads)

Attribute	Description	Default
data-policy	Specifies how the local cache handles data for a region. This setting controls behavior such as local data storage and region initialization. Note: 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. You can specify the following data policies: 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 subscription-attributes interest-policy to all. normal: Data used locally (accessed with get, stored with put, etc.) is stored in the local cache. This policy allows the contents in the cache to differ from other caches. partition: Data is partitioned across local and remote caches using the automatic data distribution behavior of partitioned regions. Additional configuration is done in the partition-attributes. 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 scope, behaving the same as for normal. persistent-partition: Behaves the same as partition and also persists data to disk. persistent-replicate: Behaves the same as replicate and also persists data to disk. preloaded: Initializes like a replicated region. After initialization, behaves like a normal region. API: setDataPolicy	normal

Attribute	Description	Default

	Example:
	
	``<region-attributes
	  data-policy="replicate">
	</region-attributes>```
	
	This is similar to using a region shortcut withrefid, however when you use the REPLICATE region shortcut, it automatically sets the region's scope to distributed-ack.
	
	``<region-attributes
	  refid="REPLICATE">
	</region-attributes>```
	
	If you use data-policy, you must set the scope explicitly.	

enable-async-conflation	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 deactivates 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. Note: To use this attribute, asynchronous queues must be enabled for slow consumers, specified with the async* gemfire properties. API: setEnableAsyncConflation Example: ```<region-attributes enable-async-conflation="false"> </region-attributes>``` 	true
enable-gateway	Determines whether the gateway is enabled for the region. When set to true, events in the region are sent to the defined gateway hubs. Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the gateway-sender-id attribute of the <region-attributes> element.	false

Attribute	Description	Default

enable-subscription-conflation	Boolean for server regions that specifies whether the server can conflate its messages to the client. A value of true enables conflation.
	
	Note: The client can override this setting with the conflate-events property in its gemfire.properties.
	
	API: setEnableSubscriptionConflation
	
	Example:
	
	`<region-attributes enable-subscription-conflation="true">`
	`</region-attributes>`
		false

gateway-sender-ids	Specifies one or more gateway sender IDs to use for distributing region events to remote Tanzu GemFire sites. Specify multiple IDs as a comma-separated list.
	
	API: addGatewaySenderId
	
	Example:
	
	`<region-attributes gateway-sender-ids="nwsender,swsender">`
	`</region-attributes>`
		not set

async-event-queue-ids	Specifies one or more asynchronous event queues to use for distributing region events an AsyncEventListener implementation, for example, for write-behind cache event handling. Specify multiple IDs as a comma-separated list.
	
	API: addAsyncEventQueueId
	
	Example:
	
	`<region-attributes async-event-queue-ids="customerqueue,ordersqueue">`
	`</region-attributes>`
		not set

hub-id	If the enable-gateway attribute is set to true, a comma-separated list of gateway hub IDs that receive events from the region. Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the gateway-sender-id attribute of the <region-attributes> element.	null

Attribute	Description	Default

id	Stores the region attribute settings in the cache with this identifier. Once stored, the attributes can be retrieved using the region attribute refid.
	
	API: setId
	
	Example:
	
	`<region-attributes id="persistent-replicated">`
	`</region-attributes>`
		not set

ignore-jta	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.
	
	API: setIgnoreJTA
	
	Example:
	
	`<region-attributes ignore-jta="true">`
	`</region-attributes>`
		false

index-update-type	Specifies whether region indexes are maintained synchronously with region modifications, or asynchronously in a background thread. In the cache.xml file, this is set as a value, asynchronous or synchronous, assigned to the index-update-type region attribute. Set this through the API by passing a boolean to the setIndexMaintenanceSynchronous method.
	
	API: setIndexMaintenanceSynchronous
	
	Example:
	
	`<region-attributes index-update-type="asynchronous">`
	`</region-attributes>`
		synchronous updates

initial-capacity	Together with the load-factor region attribute, sets the initial parameters on the underlying java.util.ConcurrentHashMap used for storing region entries.
	
	API: setInitialCapacity
	
	Example:
	
	`<region-attributes initial-capacity="20">`
	`</region-attributes>`
		16

Attribute	Description	Default

is-lock-grantor	Boolean that 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 is-lock-grantor set to true. This attribute is only relevant for regions with global scope, as only they allow locking. It affects implicit and explicit locking.
	
	API: setLockGrantor
	
	Example:
	
	`<region-attributes`
	`  is-lock-grantor="true">`
	`</region-attributes>`
		false

load-factor	Together with the initial-capacity region attribute, sets the initial parameters on the underlying java.util.ConcurrentHashMap used for storing region entries. This must be a floating point number from 0 to 1, inclusive.
	
	Note: Before you set this attribute, VMware recommends that you read the discussion of initial capacity and load factor, then see the Java API documentation for java.util.ConcurrentHashMap.
	
	API: setLoadFactor
	
	Example:
	
	`<region-attributes`
	`  load-factor="0.85">`
	`</region-attributes>`
		.75

mirror-type	Deprecated	

Attribute	Description	Default

pool-name	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.
	
	API: setPoolName
	
	Examples:
	
	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:
	
	`<client-cache>`
	`  <pool name="DatabasePool"`
	`  subscription-enabled="true">`
	`  ...`
	`  </pool>`
	`  <pool >`
	`  name="OtherPool"`
	`  subscription-enabled="true">`
	`  ...`
	`  </pool>`
	`  <region ...`
	`  <region-attributes`
	`  pool-name="DatabasePool">`
	`  </region-attributes>`
	`  ...`
	
	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:
	
	`<client-cache>`
	`  <pool`
	`  name="publisher"`
	`  subscription-enabled="true">`
	`  ...`
	`  </pool>`
	`  <region`
	`  name="myRegion"`
	`  refid="CACHING_PROXY">`
	`  </region>`
	`</client-cache>`
		not set

Attribute	Description	Default

disk-store-name	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 disk-synchronous region attribute specifies whether writes are done synchronously or asynchronously.
	API: setDiskStoreName
	Example:
	`<region-attributes`
	`  disk-store-name="myStoreA" >`
	`</region-attributes>`
		null

disk-synchronous	For regions that write to disk, boolean that specifies whether disk writes are done synchronously for the region.
	API: setDiskSynchronous
	Example:
	`<region-attributes`
	`  disk-store-name="myStoreA"`
	`  disk-synchronous="true">`
	`</region-attributes>`
		true

refid	Retrieves region shortcuts and user-defined named region attributes for attributes initialization
	API: setRefId
	Example:
	`<region-attributes`
	`  refid="persistent-replicated">`
	`  <!-- Override any stored`
	`  attribute settings that you`
	`  need to ... -->`
	`</region-attributes>`
		not set

Attribute	Description	Default

scope	Determines how updates to region entries are distributed to the other caches in the cluster 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.
	
	Note: You can configure the most common of these options with region shortcuts in RegionShortcut and ClientRegionShortcut.
	
	Note: Server regions that are not partitioned must be replicated with distributed-ack or global scope. The region shortcuts that specify REPLICATE have distributed-ack scope.
	
	Set one of the following scope values:
	
	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 cluster.
	
	global: Events are distributed to remote caches with global locking to ensure distributed cache consistency.
	
	API: setScope
	
	Example:
	
	`<region-attributes scope="distributed-ack">`
	`</region-attributes>`
		distributed-no-ack

Attribute	Description	Default

statistics-enabled	Boolean specifying whether to gather statistics on the region. Must be set to true to use expiration on the region. Tanzu GemFire provides a standard set of statistics for cached regions and region entries, which provide information for fine-tuning your cluster. Unlike other Tanzu GemFire statistics, statistics for local and distributed regions are not archived and cannot be charted. They are kept in instances of org.apache.geode.cache.CacheStatistics and made available through the region and its entries through the Region.getStatistics and Region.Entry.getStatistics methods.
	
	API: setStatisticsEnabled
	
	Example:
	
	`<region-attributes statistics-enabled="true">`
	`</region-attributes>`
		false

cloning-enabled	Boolean that determines how fromDelta applies deltas to the local cache for delta propagation. When set to true, the updates are applied to a clone of the value and then the clone is saved to the cache. When set to false, the value is modified in place in the cache.
	
	API: setCloningEnabled
	
	Example:
	
	`<region-attributes cloning-enabled="true">`
	`</region-attributes>`
		false

concurrency-checks-enabled	Boolean that determines whether members perform checks to provide consistent handling for concurrent or out-of-order updates to distributed regions. See Consistency for Region Updates.
	
	Note: Applications that use a client-cache may benefit from deactivating concurrency checking to see all events for a region. Tanzu 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 unsynchronized with the server cache.
	
	API: setConcurrencyChecksEnabled
	
	Example:
	
	`<region-attributes concurrency-checks-enabled="true">`
	`</region-attributes>`
		true

Attribute	Description	Default

off-heap	Boolean that determine whether the region uses off-heap memory to store entry values, including values for region entries and queue entries. The region continues to use heap memory for everything else, such as entry keys and the ConcurrentHashMap.
	
	API: setOffHeap
	
	Example:
	
	`<region-attributes off-heap="true">`
	`</region-attributes>`
		false

---

index

<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 for more information about indexes.

Default:

API: org.apache.geode.cache.query.QueryService.createIndex, createKeyIndex, createHashIndex

<index> Attributes

Attribute	Description	Default
name	Required. Name of the index.	
from-clause	Specifies the collections 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	Boolean that determines whether the index should be a key index. If set to 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>
```

---

entry

<entry>

An <entry> element describes an entry to be added to a region. If an entry with the given key already exists in the region, it will be replaced.

Default:

API: org.apache.geode.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

<key>

Required. Describes the key in a region entry. A key can contain either a <string> or a <declarable> sub-element.

---

string

<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

<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.

API: Declarable

Example:

```
<parameter name="cacheserver">
  <declarable>
    <class-name>org.apache.geode.addon.cache.CacheServerInitializer</class-name>
    <parameter name="system.property.prefix">
      <string>cacheserver</string>
    </parameter>
  </declarable>
</parameter>
```

---

value

<value>

Required. Describes the value of a region entry. A <value> can contain either a <string> or a <declarable> sub-element.

---

string

<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

<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.

API: Declarable

Example:

```
<parameter name="cacheserver">
  <declarable>
    <class-name>org.apache.geode.addon.cache.CacheServerInitializer</class-name>
    <parameter name="system.property.prefix">
      <string>cacheserver</string>
    </parameter>
  </declarable>
</parameter>
```

---

region

<region>

Defines a region in the cache. See <region-attributes> for more details on configuring regions. You can specify zero or more subregions within a region. See Create and Access Data Subregions for restrictions on creating subregions. For example, you cannot create a partitioned subregion.

Default:

API: org.apache.geode.cache.RegionFactory or org.apache.geode.cache.ClientRegionFactory

<region> Attributes

Attribute	Description	Default
name	Specify the name for the region. See Region Management for details.	
refid	Used to apply predefined attributes to the region being defined. If the nested region-attributes element has its own refid, the refid on the region is ignored. The refid region attribute can be set to the name of a RegionShortcut or a ClientRegionShortcut. For more information, see Region Shortcuts and Custom Named Region Attributes and Storing and Retrieving Region Shortcuts and Custom Named Region Attributes.	

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.

---

function-service

<function-service>

Configures the behavior of the function execution service.

Example:

```
<client-cache>
  ...
  <function-service>
    <function>
      <class-name>com.myCompany.tradeService.cache.func.TradeCalc</class-name>
    </function>
  </function-service>
  ...
</client-cache>
```

---

function

<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>.

Default:

API: org.apache.geode.cache.execute.FunctionService

Example:

```
<function>
  <class-name>
    com.myCompany.tradeService.cache.func.TradeCalc
  </class-name>
</function>
```

---

resource-manager

<resource-manager>

A memory monitor that tracks cache size as a percentage of total heap or off-heap memory and controls size by restricting access to the cache and prompting eviction of old entries from the cache. For tenured heap, used in conjunction with settings for JVM memory and Java garbage collection. For off-heap memory, used with the off-heap memory manager.

API: org.apache.geode.cache.control.ResourceManager

<resource-manager> 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. 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 setCriticalHeapPercentage() for details.	0
eviction-heap-percentage	Set the percentage of heap at or above which the eviction should begin on Regions configured for HeapLRU eviction. Changing this value may cause eviction to begin immediately.	• 0, if no region is configured with heap eviction • If critical-heap-percentage is set to a non-zero value, 5% less than that value. • 80%, if critical-heap-percentage is not configured.
critical-off-heap-percentage	Percentage of off-heap memory 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-off-heap-percentage	Set the percentage of off-heap memory at or above which the eviction should begin on Regions configured for HeapLRU eviction.	• 0, if no region is configured with heap eviction • If critical-off-heap-percentage is set to a non-zero value, 5% less than that value. • 80%, if critical-off-heap-percentage is not configured.

Example:

```
<client-cache>
  ...
  <resource-manager
    critical-heap-percentage="99.9"
    eviction-heap=-percentage="85"/>
  ...
</client-cache>
```

---

serialization-registration

<serialization-registration>

Set of serializer or instantiator tags to register customer DataSerializer extensions or DataSerializable implementations respectively.

Example:

```
<serialization-registration>
  <instantiator id="30">
    <class-name>com.package.MyClass</class-name>
  </instantiator>
</serialization-registration>
```

---

serializer

<serializer>

Allows you to configure the DataSerializer for this Tanzu GemFire member. It registers a custom class which extends DataSerializer to support custom serialization of non-modifiable object types inside Tanzu 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 org.apache.geode.DataSerializer.register API. Use the org.apache.geode.Instantiator API to register a DataSerializable implementation.

---

instantiator

<instantiator>

An Instantiator registers a custom class which implements the DataSerializable interface to support custom object serialization inside Tanzu 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 <client-cache>. Use the org.apache.geode.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>.

<instantiator> Attributes

Attribute	Description	Default
id	Required. ID that the Instantiator should associate with the DataSerializable type.	

---

initializer

<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>.

Default:

API: Declarable

Example:

```
<initializer>
  <class-name>MyInitializer</class-name>
  <parameter name="members">
    <string>2</string>
  </parameter>
</initializer>
```

Region Shortcuts

This topic describes the region shortcuts that you can use to configure VMware Tanzu GemFire regions.

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/>
  </region-attributes>
</region>
```

You can also create your own, named region shortcuts for common custom configurations. See Region Shortcuts and Custom Named Region Attributes.

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
```

> 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 Region Management.

For more information about using the various types of Tanzu GemFire regions and when to use them, see Region Types.

For a quick reference listing all region shortcuts, see Region Shortcuts Quick Reference.

Region Shortcuts Reference

This topic describes the region shortcuts that you can use to configure VMware Tanzu 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.

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
```

XML Definition of LOCAL

```
<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.

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
```

XML Definition of LOCAL_HEAP_LRU

```
<region name="myLHLregion">
  <region-attributes
    scope="local">
    <eviction-attributes>
      <lru-heap-percentage
        action="local-destroy">
        <class-name>
          org.apache.geode.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.

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
```

XML Definition of LOCAL_OVERFLOW

```
<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.

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
```

XML Definition of LOCAL_PERSISTENT

```
<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.

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
```

XML Definition of LOCAL_PERSISTENT_OVERFLOW

```
<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 is partitioned across each peer member that created the region.

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
```

XML Definition of PARTITION

```
<region name="myPregion">
  <region-attributes
    data-policy="partition"/>
</region>
```

PARTITION_HEAP_LRU

Description

A region configured with PARTITION_HEAP_LRU 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.

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
```

XML Definition of PARTITION_HEAP_LRU

```
<region name="myPHLregion">
  <region-attributes
    data-policy="partition">
    <eviction-attributes>
      <lru-heap-percentage
        action="local-destroy">
        <class-name>org.apache.geode.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 state that is partitioned across each peer member that creates the region. The region moves the values of entries to disk when it detects that the JVM is running low on memory.

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
```

XML Definition of PARTITION_OVERFLOW

```
<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 region shortcut has state that is partitioned across each peer member that creates the region. The region writes its state to disk and can recover that state when the member restarts.

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
```

XML Definition of PARTITION_PERSISTENT

```
<region name="myPPregion">
  <region-attributes
    data-policy="persistent-partition"/>
</region>
```

PARTITION_PERSISTENT_OVERFLOW

Description

A region configured with the PARTITION_PERSISTENT_OVERFLOW region shortcut has state that is partitioned across each peer member that creates 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.

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
```

XML Definition of PARTITION_PERSISTENT_OVERFLOW

```
<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.

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
```

XML Definition of PARTITION_PROXY

```
<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.

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
```

XML Definition of PARTITION_PROXY_REDUNDANT

```
<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 state that is partitioned across each peer member that creates the region. The region maintains an extra copy of the data in memory.

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
```

XML Definition of PARTITION_REDUNDANT

```
<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 state that is partitioned across each peer member that creates 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.

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
```

XML Definition of PARTITION_REDUNDANT_HEAP_LRU

```
<region name="myPRHLregion">
  <region-attributes
    data-policy="partition">
    <partition-attributes redundant-copies="1"/>
    <eviction-attributes>
      <lru-heap-percentage action="local-destroy">
        <class-name>
          org.apache.geode.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 state that is partitioned across each peer member that creates 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.

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
```

XML Definition of PARTITION_REDUNDANT_OVERFLOW

```
<region name="myPROregion">
  <region-attributes
    data-policy="partition">
    <partition-attributes
      redundant-copies="1"/>
    <eviction-attributes>
      <lru-heap-percentage
        action="overflow-to-disk"/>
    </eviction-attributes>
  </region-attributes>
</region>
```

PARTITION_REDUNDANT_PERSISTENT

Description

A region configured with the PARTITION_REDUNDANT_PERSISTENT region shortcut has state that is partitioned across each peer member that creates 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.

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
```

XML Definition of PARTITION_REDUNDANT_PERSISTENT

```
<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 state that is partitioned across each peer member that creates 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.

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
```

XML Definition of PARTITION_REDUNDANT_PERSISTENT_OVERFLOW

```
<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 synchronized with all other peer regions configured with a REPLICATE data policy.

Default Attributes

Region Attributes

scope:	distributed-ack

data-policy:	REPLICATE

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=myRregion --type=REPLICATE
Member  | Status
------- | ---------------------------------------
server1 | Region "/myRregion" created on "server1"
server2 | Region "/myRregion" created on "server2"
```

XML Definition of REPLICATE

```
<region name="myRregion">
  <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 synchronized 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 synchronized by registering interest in the keys of its peer's replicated regions.

Default Attributes

Region Attributes

scope:	distributed-ack
data-policy:	PRELOADED

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 synchronized by registering interest in the keys of its peer's replicated regions.

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
```

XML Definition of REPLICATE_HEAP_LRU

```
<region name="myRHLregion">
  <region-attributes
    scope="distributed-ack"
    data-policy="preloaded">
    <subscription-attributes
      interest-policy="all"/>
    <eviction-attributes>
      <lru-heap-percentage
        action="local-destroy">
        <class-name>org.apache.geode.internal.size.SizeClassOnceObjectSizer</class-name>
      </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 synchronized with all other peer regions configured with a REPLICATE data policy.

Default Attributes

Region Attributes

scope:	distributed-ack
data-policy:	REPLICATE

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
```

XML Definition of REPLICATE_OVERFLOW

```
<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 synchronized 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.

Default Attributes

Region Attributes

scope:	distributed-ack
data-policy:	PERSISTENT_REPLICATE

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
```

XML Definition of REPLICATE_PERSISTENT

```
<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 synchronized 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.

Default Attributes

Region Attributes

scope:	distributed-ack
data-policy:	PERSISTENT_REPLICATE

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
```

XML Definition of REPLICATE_PERSISTENT_OVERFLOW

```
<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.

Default Attributes

Region Attributes

scope:	distributed-ack
data-policy:	EMPTY

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
```

XML Definition of REPLICATE_PROXY

```
<region name="myRPXYregion">
  <region-attributes
    scope="distributed-ack"
    data-policy="empty"/>
</region>
```

Region Shortcuts Reference Table

This table provides a quick reference for all VMware Tanzu GemFire region shortcuts.

Region Shortcuts Default Configurations summarizes the default configurations for each region shortcut. See the cross reference for additional information about each shortcut.

Table 1. Region Shortcuts Default Configurations

Region Shortcut	Region Attributes	Other Attributes
LOCAL	scope: local data-policy: NORMAL	
LOCAL_HEAP_LRU	scope: local data-policy: NORMAL	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: local-destroy

Region Shortcut	Region Attributes	Other Attributes
LOCAL_OVERFLOW	scope: local data-policy: NORMAL	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk
LOCAL_PERSISTENT	scope: local data-policy: PERSISTENT_REPLICATE	
LOCAL_PERSISTENT_OVERFLOW	scope: local data-policy: PERSISTENT_REPLICATE	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk
PARTITION	data-policy: PARTITION	
PARTITION_HEAP_LRU	data-policy: PARTITION	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: local-destroy
PARTITION_OVERFLOW	data-policy: PARTITION	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk
PARTITION_PERSISTENT	data-policy: PERSISTENT_PARTITION	
PARTITION_PERSISTENT_OVERFLOW	data-policy: PERSISTENT_PARTITION	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk
PARTITION_PROXY	data-policy: PARTITION	Partition Attributes local-max-memory: 0
PARTITION_PROXY_REDUNDANT	data-policy: PARTITION	Partition Attributes local-max-memory: 0 redundant-copies: 1
PARTITION_REDUNDANT	data-policy: PARTITION	Partition Attributes redundant-copies: 1
PARTITION_REDUNDANT_HEAP_LRU	data-policy: PARTITION	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: local-destroy Partition Attributes redundant-copies: 1

Region Shortcut	Region Attributes	Other Attributes
PARTITION_REDUNDANT_OVERFLOW	data-policy: PARTITION	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk Partition Attributes redundant-copies: 1
PARTITION_REDUNDANT_PERSISTENT	data-policy: PERSISTENT_PARTITION	Partition Attributes redundant-copies: 1
PARTITION_REDUNDANT_PERSISTENT_OVERFLOW	data-policy: PERSISTENT_PARTITION	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk Partition Attributes redundant-copies: 1
REPLICATE	data-policy: REPLICATE scope: distributed-ack	
REPLICATE_HEAP_LRU	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 synchronized by registering interest in the keys of its peer's replicated regions. Subscription Attributes interest-policy: all
REPLICATE_OVERFLOW	data-policy: REPLICATE scope: distributed-ack	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk
REPLICATE_PERSISTENT	data-policy: PERSISTENT_REPLICATE scope: distributed-ack	
REPLICATE_PERSISTENT_OVERFLOW	data-policy: PERSISTENT_REPLICATE scope: distributed-ack	Eviction Attributes eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk

Region Shortcut	Region Attributes	Other Attributes
REPLICATE_PROXY	data-policy: EMPTY scope: distributed-ack	

Exceptions and System Failures

This topic describes how your application must catch certain classes to handle all of the exceptions and system failures that can be thrown by VMware Tanzu 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 Tanzu 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 Tanzu GemFire version of java.lang.RuntimeException. You can also look at the method details in the Region API javadocs for Tanzu GemFire exceptions you may want to catch.
- SystemFailure. In addition to exception management, Tanzu GemFire provides a class to help you manage catastrophic failure in your cluster, 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 org.apache.geode package.

To see the exceptions thrown by a specific method, refer to the method's online Java documentation.

A Tanzu GemFire system member can also throw exceptions generated by third-party software such as java.lang classes. For assistance in handling these exceptions, see the vendor documentation.

Memory Requirements for Cached Data

This topic discusses memory requirements for cached data when using VMware Tanzu GemFire.

Tanzu GemFire solutions architects must 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 Tanzu 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 Tanzu GemFire Data Region Design

The following guidelines apply to 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.

Memory calculation about keys and entries (objects) and region overhead for them can be divided by the number of members of the cluster 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 Tanzu GemFire is per member.

For each entry added to a region, the Tanzu 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 to object references and headers using more memory.

There are several additional considerations for calculating your memory requirements:

- Size of your stored data. To estimate the size of your stored data, determine first whether you are storing the data in serialized or non-serialized form. In general, the non-serialized form will be the larger of the two. See Determining Object Serialization Overhead

  Objects in Tanzu 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, Tanzu 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. When calculating application overhead, 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.

  The following section “Calculating Application Object Overhead” provides details on how to estimate the memory overhead of the keys and values stored in the cache.

Calculating Application Object Overhead

To compute the memory overhead of a Java object, perform the following steps:

1. Determine the object header size. Each Java object has an object header. For a 32-bit JVM, it is 8 bytes. For a 64-bit JVM with a heap less than or equal to 32 GB, it is 12 bytes. For a 64-bit JVM with a heap greater than 32 GB, it is 16 bytes.
2. Determine the memory overhead of the fields of the object. For every instance field (including fields from super classes), add in the field's size. For primitive fields the sizes are:
    - 8 for long and double
    - 4 for int and float
    - 2 for char and short
    - 1 for byte and boolean

    For object reference fields, the size is 8 bytes for 64-bit JVM with a heap greater than 32 GB in size or a JVM with ZGC activated. For all other JVMs, use 4 bytes.
3. Add up the numbers from Step 1 and 2 and round it up to the next multiple of 8. The result is the memory overhead of that Java object.

Java arrays. To compute the memory overhead of a Java array, you would add the object header (since the array is an object) and a primitive int field that contains its size. Treat each element of the array as if it was an instance field. For example, a byte array of the size 100 bytes would have one object header, one int field, and 100 byte fields. Use the three step process described above to do the computation.

Serialized objects. When computing the memory overhead of a serialized value, remember that the serialized form is stored in a byte array. Therefore, to figure out how many bytes the serialized form contains, compute the memory overhead of a Java byte array of that size and then add in the size of the serialized value wrapper.

When a value is initially stored in the cache in serialized form, a wrapper around the value is introduced that is kept in memory for the life of that value even if the value is later deserialized. Although this wrapper is only used internally, it does add to the memory footprint. The wrapper is an object with one int field and one object reference.

If you are using partitioned regions, every value is initially stored in serialized form. For other region types only values that come from a remote member (peers or clients) are initially stored in serialized form. (This is the most common case.) However, if a local operation stores the value in the local JVM's cache, then the value will be stored in object form. A large number of operations can cause a value stored in serialized form to be deserialized. Any operation that needs the object form of the value to be local can cause this deserialization. If such operations are performed, then that value will be stored in object form (with the additional serialized wrapper) and the serialized form becomes garbage.

> An exception to this is if the serialized form is encoded with PDX, then setting read-serialized to true will keep the serialized form in the cache.

See Determining Object Serialization Overhead for additional information about how to calculate memory usage requirements for storing serialized objects.

Using Key Storage Optimization

Keys are stored in object form except for certain classes where the storage of keys is optimized. Key storage is optimized by replacing the entry's object reference to the key with one or two primitive fields on the entry that store the key's data "inline". The following rules apply to determine whether a key is stored "inline":

- If the key's class is java.lang.Integer, java.lang.Long, or java.util.UUID, then the key is always stored inline. The memory overhead for an inlined Integer or Long key is 0 (zero). The memory overhead for an inlined UUID is 8.
- If the key's class is java.lang.String, then the key will be inlined if the string's length is small enough.
    - For ASCII strings whose length is less than 8, the inline memory overhead is 0 (zero).
    - For ASCII strings whose length is less than 16, the inline memory overhead is 8.
    - For non-ASCII strings whose length is less then 4, the inline memory overhead is 0 (zero).
    - For non-ASCII strings whose length is less then 8 the inline memory overhead is 8.

  All other strings are not inlined.

When to deactivate inline key storage. In some cases, storing keys inline may introduce extra memory or CPU usage. If all of your keys are also referenced from some other object, then it is better to not inline the key. If you frequently ask for the key from the region, then you may want to keep the object form stored in the cache so that you do not need to recreate the object form constantly. Note that the basic operation of checking whether a key is in a region does not require the object form but uses the inline primitive data.

The key inlining feature can be deactivated by specifying the following Tanzu GemFire property upon member startup:

```
-Dgemfire.DISABLE_INLINE_REGION_KEYS=true
```

Measuring Cache Overhead

For heap sizes smaller than 32 GB, 64-bit JVMs using the CMS or G1 garbage collector can benefit from Java's -XX:+UseCompressedOops option, which significantly reduces Java's heap usage. Java enables this option by default on JVMs that can benefit from it. JVMs using ZGC and JVMs with heap sizes of 32 GB or larger cannot use this option.

Actual heap usage varies based on a number of factors, including the JVM type and the platform that you run on. For JVMs not using compressed oops, heap usage will usually be larger than with compressed oops and may be as much as 80% higher.

This table estimates the cache overhead in a 64-bit JVM. The overhead is required even when an entry is overflowed or persisted to disk.

> Memory consumption for object headers and object references can vary for 64-bit JVMs, different JVM implementations, and different JDK versions.

Region Attribute	Compressed Oops	No Compressed Oops
Each entry	+64 bytes	+90 bytes
Concurrency checking deactivated	-16 bytes	-24 bytes
Statistics activated	+16 bytes	+16 bytes
Persisted	+132 bytes	+166 bytes
Overflow only	+56 bytes	+72 bytes
LRU eviction controller	+16 bytes	+24 bytes

Region Attribute	Compressed Oops	No Compressed Oops
Global scope	+100 bytes	+142 bytes
Expiration configured	+112 bytes	+152 bytes
Optional user attribute	+44 bytes	+64 bytes

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:

- Single unique value per region entry for the indexed expression: The index introduces approximately 117 bytes with Compressed Oops or approximately 167 bytes with No Compressed Oops. An example of this type of index is: fromClause="/portfolios", indexedExpression="id".
- Multiple values for the indexed expression, but no two entries have the same value: The index introduces a maximum of:
    - 290c + 290 bytes per region entry with Compressed Oops
    - 396c + 396 bytes per region entry with No Compressed Oops

  Where c is the average number of values per region entry for the expression.

Estimating Management and Monitoring Overhead

The Tanzu GemFire JMX management and monitoring system contributes to memory overhead and should be accounted for when establishing the memory requirements for your deployment. Specifically, the memory footprint of any processes (such as locators) that are running as JMX managers can increase.

For each resource in the cluster that is being managed and monitored by the JMX Manager (for example, each MXBean such as MemberMXBean, RegionMXBean, DiskStoreMXBean, LockServiceMXBean and so on), you should add 10 KB of required memory to the JMX Manager node.

Determining Object Serialization Overhead

Tanzu GemFire PDX serialization can provide significant space savings over Java Serializable in addition to better performance. In some cases we have seen savings of up to 65%, but the savings will vary depending on the domain objects. PDX serialization is most likely to provide the most space savings of all available options. DataSerializable is more compact, but it requires that objects are deserialized on access, so that should be taken into account. On the other hand, PDX serializable does not require deserialization for most operations, and because of that, it may provide greater space savings.

In any case, the kinds and volumes of operations that would be done on the server side should be considered in the context of data serialization, as Tanzu GemFire has to deserialize data for some types of operations (access). For example, if a function invokes a get operation on the server side, the value returned from the get operation will be deserialized in most cases (the only time it will not be deserialized is when PDX serialization is used and the read-serialized attribute is set). The only way to find out the actual overhead is by running tests, and examining the memory usage.

Some additional serialization guidelines and tips:

- If you are using compound objects, do not mix using standard Java serialization with Tanzu GemFire serialization (either DataSerializable or PDX). Standard Java serialization functions correctly when mixed with Tanzu GemFire serialization, but it can end up producing many more serialized bytes.

To determine if you are using standard Java serialization, specify the -DDataSerializer.DUMP_SERIALIZED=true upon process execution. Then check your log for messages of this form:

```
DataSerializer Serializing an instance of <className>
```

Any classes list are being serialized with standard Java serialization. You can optimize your serialization by handling those classes in a PdxSerializer or a DataSerializer or changing the class to be PdxSerializable or DataSerializable.

- One way to determine the serialized size of an object is to create an instance of that object and then call DataSerializer.writeObject(obj dataOutput) where “dataOutput” wraps a ByteArrayOutputStream. You can then ask the stream for its size, and it will return the serialized size. Make sure you have configured your PdxSerializer and DataSerializers configured before you call writeObject.

If you want to estimate memory usage for PDX serialized data, the following table provides estimated sizes for various types when using PDX serialization:

Type	Memory Usage
boolean	1 byte
byte	1 byte
char	2 bytes
short	2 bytes
int	4 bytes
long	8 bytes
float	8 bytes
String	String.length + 3 bytes
Domain Object	9 bytes (for PDX header) + object serialization length (total all member fields) + 1 to 4 extra bytes (depends on the total size of Domain object)

A note of caution: If the domain object contains many domain objects as member fields, then the memory overhead of PDX serialization can be considerably more than other types of serialization.

Calculating Socket Memory Requirements

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 Tanzu GemFire connection property *conserve-sockets* is set to true, these threads use the already-established peer connections for this communication.

If *conserve-sockets* is false (the default), 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 large amount of memory to be consumed by sockets.

> There is also JVM overhead for the thread stack for each client connection being processed, set at 256KB or 512KB for most JVMs . On some JVMs you can reduce it to 128KB. You can use the Tanzu GemFire max-threads property or the Tanzu 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.

Connections	Memory requirements
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 cluster 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 Note: It is = 2* current number of clients connected * number of peers. Each connection spawns a thread.
Subscription Queues	
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/or 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 +
Tanzu GemFire classes and JVM overhead	Roughly 50MB
Thread overhead	

Connections	Memory requirements
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.)

GemFire Statistics List

This topic describes the primary statistics gathered by VMware Tanzu GemFire when statistics are enabled.

All statistics gathering requires the gemfire.properties statistic-sampling-enabled in gemfire.properties file to be true. The statistic-sampling-enabled property is deprecated as it will be enabled by default. It will be removed in upcoming releases. Statistics that use time require the gemfire.properties enable-time-statistics to be true.

Performance statistics are collected for each Java application or cache server that connects to a cluster.

- Cache Performance (CachePerfStats)
- Cache Server (CacheServerStats)
- Client-Side Notifications (CacheClientUpdaterStats)
- Client-to-Server Messaging Performance (ClientStats)
- Client Connection Pool (PoolStats)
- Continuous Querying (CqQueryStats)
- Disk Space Usage (DiskDirStatistics)
- Disk Store Statistics (DiskStoreStatistics)
- Disk Usage and Performance (DiskRegionStatistics)
- Distributed System Messaging (DistributionStats)
- Distributed Lock Services (DLockStats)
- Function Execution (FunctionServiceStatistics)
- Gateway Queue (GatewaySenderStatistics)
- Indexes (IndexStats)
- JVM Performance
- Locator (LocatorStats)
- Lucene Indexes (LuceneIndexStats)
- Off-Heap (OffHeapMemoryStats)
- Operating System Statistics - Linux
- Partitioned Regions (PartitionedRegion<partitioned_region_name>Statistics)
- Region Entry Eviction – Count-Based (LRUStatistics)
- Region Entry Eviction - Heap-based eviction (HeapLRUStatistics)
- Region Entry Eviction – Size-based (MemLRUStatistics)
- Server Notifications for All Clients (CacheClientNotifierStatistics)
- Server Notifications for Single Client (CacheClientProxyStatistics)

- Server-to-Client Messaging Performance (ClientSubscriptionStats)
- Statistics Collection (StatSampler)

Cache Performance (CachePerfStats)

Statistics for the Tanzu GemFire cache. These can be used to determine the type and number of cache operations being performed and how much time they consume.

Regarding Tanzu GemFire cache transactions, transaction-related 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
cacheListenerCallTime	Total time spent doing cache listener calls.
cacheListenerCallsCompleted	Total number of times a cache listener call has completed.
cacheListenerCallsInProgress	Current number of threads doing a cache listener call.
cacheWriterCallTime	Total time spent doing cache writer calls.
cacheWriterCallsCompleted	Total number of times a cache writer call has completed.
cacheWriterCallsInProgress	Current number of threads doing a cache writer call.
clearTimeouts	Number of timeouts waiting for events concurrent to a clear() operation to be received and applied before performing the clear()
clears	The total number of times a clear has been done on this cache.
compressTime	The total time spent compressing data.
compressions	The total number of compression operations.
conflatedEvents	Number of events not delivered due to conflation. Typically this means that the event arrived after a later event was already applied to the cache.
creates	The total number of times an entry is added to this cache.
decompressTime	The total time spent decompressing data.
decompressions	The total number of decompression operations.
deltaFailedUpdates	The total number of times entries in this cache failed to be updated through delta bytes.
deltaFullValuesRequested	The total number of times a full value was requested by this cache.
deltaFullValuesSent	The total number of times a full value was sent to a remote cache.
deltaGetInitialImagesCompleted	Total number of times delta getInitialImages initiated by this cache have completed.
deltaUpdates	The total number of times entries in this cache are updated through delta bytes.

Statistic	Description
deltaUpdatesTime	Total time spent applying the received delta bytes to entries in this cache.
deltasPrepared	The total number of times delta was prepared in this cache.
deltasPreparedTime	Total time spent preparing delta bytes in this cache.
deltasSent	The total number of times delta was sent to remote caches. This excludes deltas sent from server to client.
destroys	The total number of times a cache object entry has been destroyed in this cache.
diskTasksWaiting	Current number of disk tasks (oplog compactions, asynchronous recoveries, etc) that are waiting for a thread to run the operation
entries	Current number of entries in the cache. This does not include any entries that are tombstones. See tombstoneCount.
eventQueueSize	The number of cache events waiting to be processed.
eventQueueThrottleCount	The total number of times a thread was delayed in adding an event to the event queue.
eventQueueThrottleTime	The total amount of time, in nanoseconds, spent delayed by the event queue throttle.
eventThreads	The number of threads currently processing events.
eventsQueued	Number of events attached to other events for callback invocation
evictByCriteria_evaluationTime	Total time taken for evaluation of user expression during eviction
evictByCriteria_evaluations	Total number of evaluations for eviction
evictByCriteria_evictionTime	Time taken for eviction process
evictByCriteria_evictions	The total number of entries evicted
evictByCriteria_evictionsInProgress	Total number of evictions in progress
evictWorkTime	Total time spent doing eviction work in background threads
evictorJobsCompleted	Number of evictor jobs completed
evictorJobsStarted	Number of evictor jobs started
evictorQueueSize	Number of jobs waiting to be picked up by evictor threads
expiryDelayTime	The total amount of time, in milliseconds, that entry expirations were delayed. The delay is the difference between when an entry was meant to expire and when it actually did expire.
expiryDelays	The total number of entries that were delayed in expiring.
expiryTasks	The current number of entries waiting to expire.
expiryThreads	The number of threads configured to process expiry tasks.
exportTime	The total time spent exporting entries into a snapshot file.
exportedEntries	The total number of entries exported into a snapshot file.

Statistic	Description
getInitialImageKeysReceived	Total number of keys received while doing getInitialImage operations.
getInitialImageTime	Total time spent doing getInitialImages for region creation.
getInitialImagesCompleted	Total number of times getInitialImages (both delta and full GII) initiated by this cache have completed.
getInitialImagesInProgress	Current number of getInitialImage operations currently in progress.
getTime	Total time spent doing get operations from this cache (including netsearch and netload)
gets	The total number of times a successful get has been done on this cache.
handlingNetsearchesCompleted	Total number of times handling a network search initiated by a remote cache completed with success.
handlingNetsearchesFailed	Total number of times handling a network search initiated by a remote cache failed without success.
handlingNetsearchesFailedTime	Total time spent handling failed network searches for remote caches.
handlingNetsearchesInProgress	Current number of threads handling a network search initiated by a remote cache.
handlingNetsearchesTime	Total time spent handling successful network searches for remote caches.
importTime	The total time spent importing entries from a snapshot file.
importedEntries	The total number of entries imported from a snapshot file.
indexInitializationCompleted	Total number of index initializations that have completed
indexInitializationInProgress	Current number of index initializations in progress
indexInitializationTime	Total amount of time spent initializing indexes
indexUpdateCompleted	Total number of ops that have completed
indexUpdateInProgress	Current number of ops in progress
indexUpdateTime	Total amount of time spent doing this op
invalidates	The total number of times an existing cache object entry value in this cache has been invalidated
loadTime	Total time spent invoking loaders on this cache.
loadsCompleted	Total number of times a load on this cache has completed (as a result of either a local get() or a remote netload).
loadsInProgress	Current number of threads in this cache doing a cache load.
metaDataRefreshCount	Total number of times the meta data is refreshed due to hopping observed.

Statistic	Description
misses	Total number of times a get on the cache did not find a value already in local memory. The number of hits (i.e. gets that did not miss) can be calculated by subtracting misses from gets.
netloadTime	Total time spent doing network loads on this cache.
netloadsCompleted	Total number of times a network load initiated on this cache has completed.
netloadsInProgress	Current number of threads doing a network load initiated by a get() in this cache.
netsearchTime	Total time spent doing network searches for cache values.
netsearchesCompleted	Total number of times network searches initiated by this cache have completed.
netsearchesInProgress	Current number of threads doing a network search initiated by a get() in this cache.
nonReplicatedTombstonesSize	Amount of memory consumed by destroyed entries in non-replicated regions
partitionedRegionQueryRetries	Total number of times an OQL Query on a Partitioned Region had to be retried
partitionedRegions	The current number of partitioned regions in the cache.
postCompressedBytes	The total number of bytes after compressing.
preCompressedBytes	The total number of bytes before compressing.
previouslySeenEvents	The number of previously seen events ignored by the event tracker.
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).
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).
putalls	The total number of times a map is added or replaced in this cache as a result of a local operation. Note that this only counts putAlls done explicitly on this cache. It does not count updates pushed from other caches.
puts	The total number of times an entry is added or replaced in this cache as a result of a local operation (put(), create(), or get() which results in load, netsearch, or netloading a value). Note that this only counts puts done explicitly on this cache. It does not count updates pushed from other caches.
queryExecutionTime	Total time spent executing queries
queryExecutions	Total number of times some query has been executed
queryResultsHashCollisionProbeTime	Total time spent probing the hashtable in an OQL result set due to hash code collisions, includes reads, writes, and rehashes
queryResultsHashCollisions	Total number of times an hash code collision occurred when inserting an object into an OQL result set or rehashing it
regions	The current number of regions in the cache.
reliableQueueMax	Maximum size in megabytes allotted for disk usage to queue for distribution to required roles.

Statistic	Description
reliableQueueSize	Current size in megabytes of disk used to queue for distribution to required roles.
reliableQueuedOps	Current number of cache operations queued 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.
reliableRegionsMissingFullAccess	Current number of regions configured for reliablity that are missing require roles with full access
reliableRegionsMissingLimitedAccess	Current number of regions configured for reliablity that are missing required roles with Limited access
reliableRegionsMissingNoAccess	Current number of regions configured for reliablity that are missing required roles with No access
reliableRegionsQueuing	Current number regions configured for reliability that are queuing for required roles.
removeAllTime	Total time spent performing removeAlls that originated in this cache. This includes time spent waiting for the removeAll to be done in remote caches (if required).
removeAlls	The total number of removeAll operations that originated in this cache. Note that this only counts removeAlls done explicitly on this cache. It does not count removes pushed from other caches.
replicatedTombstonesSize	Amount of memory consumed by destroyed entries in replicated or partitioned regions
retries	Number of times a concurrent destroy followed by a create has caused an entry operation to need to retry.
stateFlushOperationTime	Total time to complete state flush operation
tombstoneGCs	Number of garbage-collections performed on destroyed entries
tombstones	Number of destroyed entries that are retained for concurrent modification detection
txCommitChanges	Total number of changes made by committed transactions.
txCommitTime	The total amount of time, in nanoseconds, spent doing successful transaction commits.
txCommits	Total number times a transaction commit has succeeded.
txConflictCheckTime	The total amount of time, in nanoseconds, spent doing conflict checks during transaction commit
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.
txFailureChanges	Total number of changes lost by failed transactions.
txFailureTime	The total amount of time, in nanoseconds, spent doing failed transaction commits.
txFailures	Total number times a transaction commit has failed.
txRollbackChanges	Total number of changes lost by explicit transaction rollbacks.
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.
txRollbackTime	The total amount of time, in nanoseconds, spent doing explicit transaction rollbacks.

Statistic	Description
txRollbacks	Total number 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.
updateTime	Total time spent performing an update.
updates	The total number of updates originating remotely that have been applied to this cache.

Cache Server (CacheServerStats)

Statistics used for cache servers and for gateway receivers are recorded in CacheServerStats in a cache server. The primary statistics are:

Statistic	Description
abandonedReadRequests	Number of read opertations abandond by clients
abandonedWriteRequests	Number of write opertations abandond by clients
acceptThreadStarts	Total number of threads created to deal with an accepted socket. Note that this is not the current number of threads.
acceptsInProgress	Current number of server accepts that are attempting to do the initial handshake with the client.
batchSize	The size (in bytes) of the batches received.
clearRegionRequests	Number of cache client clearRegion requests.
clearRegionResponses	Number of clearRegion responses written to the cache client.
clientNotificationRequests	Number of cache client notification requests.
clientReadyRequests	Number of cache client ready requests.
clientReadyResponses	Number of client ready responses written to the cache client.
clientRequestsInProgress	Current number of client requests in progress.
closeConnectionRequests	Number of cache client close connection requests.
connectionLoad	The load from client to server connections as reported by the load probe installed in this server
connectionThreadStarts	Total number of threads created to deal with a client connection. Note that this is not the current number of threads.
connectionThreads	Current number of threads dealing with a client connection.
connectionsTimedOut	Total number of connections that have been timed out by the server because of client inactivity
containsKeyRequests	Number of cache client containsKey requests.
containsKeyResponses	Number of containsKey responses written to the cache client.
currentClientConnections	Number of sockets accepted and used for client to server messaging.
currentClients	Number of client virtual machines connected.
currentQueueConnections	Number of sockets accepted and used for server to client messaging.
destroyRegionRequests	Number of cache client destroyRegion requests.
destroyRegionResponses	Number of destroyRegion responses written to the cache client.

Statistic	Description
destroyRequests	Number of cache client destroy requests.
destroyResponses	Number of destroy responses written to the cache client.
failedConnectionAttempts	Number of failed connection attempts.
getAllRequests	Number of cache client getAll requests.
getAllResponses	Number of getAll responses written to the cache client.
getRequests	Number of cache client get requests.
getResponses	Number of get responses written to the cache client.
invalidateRequests	Number of cache client invalidate requests.
invalidateResponses	Number of invalidate responses written to the cache client.
loadPerConnection	The estimate of how much load is added for each new connection as reported by the load probe installed in this server
loadPerQueue	The estimate of how much load is added for each new connection as reported by the load probe installed in this server
messageBytesBeingReceived	Current number of bytes consumed by messages being received or processed.
messagesBeingReceived	Current number of message being received off the network or being processed after reception.
outOfOrderGatewayBatchIds	Number of Out of order batch IDs.
processBatchRequests	Number of cache client processBatch requests.
processBatchResponses	Number of processBatch responses written to the cache client.
processBatchTime	Total time spent in processing a cache client processBatch request.
processClearRegionTime	Total time spent in processing a cache client clearRegion request, including the time to clear the region from the cache.
processClientNotificationTime	Total time spent in processing a cache client notification request.
processClientReadyTime	Total time spent in processing a cache client ready request, including the time to destroy an object from the cache.
processCloseConnectionTime	Total time spent in processing a cache client close connection request.
processContainsKeyTime	Total time spent processing a containsKey request.
processDestroyRegionTime	Total time spent in processing a cache client destroyRegion request, including the time to destroy the region from the cache.
processDestroyTime	Total time spent in processing a cache client destroy request, including the time to destroy an object from the cache.
processGetAllTime	Total time spent in processing a cache client getAll request.
processGetTime	Total time spent in processing a cache client get request, including the time to get an object from the cache.
processInvalidateTime	Total time spent in processing a cache client invalidate request, including the time to invalidate an object from the cache.
processPutAllTime	Total time spent in processing a cache client putAll request, including the time to put all objects into the cache.

Statistic	Description
processPutTime	Total time spent in processing a cache client put request, including the time to put an object into the cache.
processQueryTime	Total time spent in processing a cache client query request, including the time to destroy an object from the cache.
processRegisterInterestRequestTime	The total time for processing the register interest command from the client.
processRemoveAllTime	Total time spent in processing a cache client removeAll request, including the time to remove all objects from the cache.
processSizeTime	Total time spent in processing a cache client size request, including the time to size an object from the cache.
processUpdateClientNotificationTime	Total time spent in processing a client notification update request.
putAllRequests	Number of cache client putAll requests.
putAllResponses	Number of putAll responses written to the cache client.
putRequests	Number of cache client put requests.
putResponses	Number of put responses written to the cache client.
queryRequests	Number of cache client query requests.
queryResponses	Number of query responses written to the cache client.
queueInitializationsInProgress	Current number of threads working on creating subscription queues on this server.
queueLoad	The load from queues as reported by the load probe installed in this server
readClearRegionRequestTime	Total time spent in reading clearRegion requests.
readClientNotificationRequestTime	Total time spent in reading client notification requests.
readClientReadyRequestTime	Total time spent in reading cache client ready requests.
readCloseConnectionRequestTime	Total time spent in reading close connection requests.
readContainsKeyRequestTime	Total time spent reading containsKey requests.
readDestroyRegionRequestTime	Total time spent in reading destroyRegion requests.
readDestroyRequestTime	Total time spent in reading destroy requests.
readGetAllRequestTime	Total time spent in reading getAll requests.
readGetRequestTime	Total time spent in reading get requests.
readInvalidateRequestTime	Total time spent in reading invalidate requests.
readProcessBatchRequestTime	Total time spent in reading processBatch requests.
readPutAllRequestTime	Total time spent in reading putAll requests.
readPutRequestTime	Total time spent in reading put requests.
readQueryRequestTime	Total time spent in reading query requests.
readRegisterInterestRequestTime	The total time for reading the register interest request from the client

Statistic	Description
readRemoveAllRequestTime	Total time spent in reading removeAll requests.
readSizeRequestTime	Total time spent in reading size requests.
readUpdateClientNotificationRequestTime	Total time spent in reading client notification update requests.
receivedBytes	Total number of bytes received from clients.
removeAllRequests	Number of cache client removeAll requests.
removeAllResponses	Number of removeAll responses written to the cache client.
sentBytes	Total number of bytes sent to clients.
sizeRequests	Number of cache client size requests.
sizeResponses	Number of size responses written to the cache client.
threadQueueSize	Current number of connections waiting for a thread to start processing their message.
updateClientNotificationRequests	Number of cache client notification update requests.
writeClearRegionResponseTime	Total time spent in writing clearRegion responses.
writeClientReadyResponseTime	Total time spent in writing client ready responses.
writeContainsKeyResponseTime	Total time spent writing containsKey responses.
writeDestroyRegionResponseTime	Total time spent in writing destroyRegion responses.
writeDestroyResponseTime	Total time spent in writing destroy responses.
writeGetAllResponseTime	Total time spent in writing getAll responses.
writeGetResponseTime	Total time spent in writing get responses.
writeInvalidateResponseTime	Total time spent in writing invalidate responses.
writeProcessBatchResponseTime	Total time spent in writing processBatch responses.
writePutAllResponseTime	Total time spent in writing putAll responses.
writePutResponseTime	Total time spent in writing put responses.
writeQueryResponseTime	Total time spent in writing query responses.
writeRegisterInterestResponseTime	The total time for sending the register interest response back to the client
writeRemoveAllResponseTime	Total time spent in writing removeAll responses.
writeSizeResponseTime	Total time spent in writing size responses.

Client-Side Notifications (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
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.
receivedBytes	Total number of bytes received from the server.

Client-to-Server Messaging Performance (ClientStats and ClientSendStats)

These statistics are in a client and they describe all the messages sent from the client to a specific server. The primary statistics of ClientStats are:

Statistic	Description
addPdxTypeFailures	Total number of addPdxType operation attempts that have failed
addPdxTypeInProgress	Current number of addPdxType operations being executed
addPdxTypeSuccessful	Total number of addPdxType operations completed successfully
addPdxTypeTime	Total amount of time, in nanoseconds spent doing addPdxType successfully/unsuccessfully.
addPdxTypeTimeouts	Total number of addPdxType operation attempts that have timed out
authentication	Total number of authentications completed successfully
authenticationFailure	Total number of authentications attempts that have failed
authenticationInProgress	Current number of authentications being executed
authenticationTime	Total amount of time, in nanoseconds spent doing authentications
authenticationTimeout	Total number of authentications attempts that have timed out
clearFailures	Total number of clear attempts that have failed
clearTime	Total amount of time, in nanoseconds spent doing clears
clearTimeouts	Total number of clear attempts that have timed out
clears	Total number of clears completed successfully
clearsInProgress	Current number of clears being executed
closeCQFailures	Total number of closeCQ attempts that have failed
closeCQTime	Total amount of time, in nanoseconds spent doing closeCQs
closeCQTimeouts	Total number of closeCQ attempts that have timed out
closeCQs	Total number of closeCQs completed successfully
closeCQsInProgress	Current number of closeCQs being executed
closeConFailures	Total number of closeCon attempts that have failed
closeConTime	Total amount of time, in nanoseconds spent doing closeCons
closeConTimeouts	Total number of closeCon attempts that have timed out
closeCons	Total number of closeCons completed successfully
closeConsInProgress	Current number of closeCons being executed
commitFailures	Total number of commit attempts that have failed

Statistic	Description
commitTime	Total amount of time, in nanoseconds spent doing commits
commitTimeouts	Total number of commit attempts that have timed out
commits	Total number of commits completed successfully
commitsInProgress	Current number of commits being executed
connections	Current number of connections
connects	Total number of times a connection has been created.
containsKeyFailures	Total number of containsKey attempts that have failed
containsKeyTime	Total amount of time, in nanoseconds spent doing containsKeys
containsKeyTimeouts	Total number of containsKey attempts that have timed out
containsKeys	Total number of containsKeys completed successfully
containsKeysInProgress	Current number of containsKeys being executed
createCQFailures	Total number of createCQ attempts that have failed
createCQTime	Total amount of time, in nanoseconds spent doing createCQs
createCQTimeouts	Total number of createCQ attempts that have timed out
createCQs	Total number of createCQs completed successfully
createCQsInProgress	Current number of createCQs being executed
destroyFailures	Total number of destroy attempts that have failed
destroyRegionFailures	Total number of destroyRegion attempts that have failed
destroyRegionTime	Total amount of time, in nanoseconds spent doing destroyRegions
destroyRegionTimeouts	Total number of destroyRegion attempts that have timed out
destroyRegions	Total number of destroyRegions completed successfully
destroyRegionsInProgress	Current number of destroyRegions being executed
destroyTime	Total amount of time, in nanoseconds spent doing destroys
destroyTimeouts	Total number of destroy attempts that have timed out
destroys	Total number of destroys completed successfully
destroysInProgress	Current number of destroys being executed
disconnects	Total number of times a connection has been destroyed.
executeFunctionFailures	Total number of Function attempts that have failed
executeFunctionTime	Total amount of time, in nanoseconds spent doing Functions
executeFunctionTimeouts	Total number of Function attempts that have timed out
executeFunctions	Total number of Functions completed successfully
executeFunctionsInProgress	Current number of Functions being executed
gatewayBatchFailures	Total number of gatewayBatch attempts that have failed
gatewayBatchTime	Total amount of time, in nanoseconds spent doing gatewayBatchs

Statistic	Description
gatewayBatchTimeouts	Total number of gatewayBatch attempts that have timed out
gatewayBatchs	Total number of gatewayBatchs completed successfully
gatewayBatchsInProgress	Current number of gatewayBatchs being executed
getAllFailures	Total number of getAll attempts that have failed
getAllTime	Total amount of time, in nanoseconds spent doing getAlls
getAllTimeouts	Total number of getAll attempts that have timed out
getAlls	Total number of getAlls completed successfully
getAllsInProgress	Current number of getAlls being executed
getClientPRMetadataFailures	Total number of getClientPRMetadata operation attempts that have failed
getClientPRMetadataInProgress	Current number of getClientPRMetadata operations being executed
getClientPRMetadataSuccessful	Total number of getClientPRMetadata operations completed successfully
getClientPRMetadataTime	Total amount of time, in nanoseconds spent doing getClientPRMetadata successfully/unsuccessfully
getClientPRMetadataTimeouts	Total number of getClientPRMetadata operation attempts that have timed out
getClientPartitionAttributesFailures	Total number of getClientPartitionAttributes operation attempts that have failed
getClientPartitionAttributesInProgress	Current number of getClientPartitionAttributes operations being executed
getClientPartitionAttributesSuccessful	Total number of getClientPartitionAttributes operations completed successfully
getClientPartitionAttributesTime	Total amount of time, in nanoseconds spent doing getClientPartitionAttributes successfully/unsuccessfully.
getClientPartitionAttributesTimeouts	Total number of getClientPartitionAttributes operation attempts that have timed out
getDurableCQs	Total number of getDurableCQs completed successfully
getDurableCQsFailures	Total number of getDurableCQs attempts that have failed
getDurableCQsInProgress	Current number of getDurableCQs being executed
getDurableCQsTime	Total amount of time, in nanoseconds spent doing getDurableCQs
getDurableCQsTimeouts	Total number of getDurableCQs attempts that have timed out
getEntryFailures	Total number of getEntry attempts that have failed
getEntryTime	Total amount of time, in nanoseconds spent doing getEntry processings
getEntryTimeouts	Total number of getEntry attempts that have timed out
getEntrys	Total number of getEntry messages completed successfully
getEntrysInProgress	Current number of getEntry messages being executed
getEventValue	Total number of getEventValue completed successfully

Statistic	Description
getEventValueFailure	Total number of getEventValue attempts that have failed
getEventValueInProgress	Current number of getEventValue being executed
getEventValueTime	Total amount of time, in nanoseconds spent doing getEventValue
getEventValueTimeout	Total number of getEventValue attempts that have timed out
getFailures	Total number of get attempts that have failed
getFunctionAttribute	Total number of getFunctionAttribute completed successfully
getFunctionAttributeFailure	Total number of getFunctionAttribute attempts that have failed
getFunctionAttributeInProgress	Current number of getFunctionAttrbute being executed
getFunctionAttributeTime	Total amount of time, in nanoseconds spent doing getFunctionAttribute
getFunctionAttributeTimeout	Total number of getFunctionAttribute attempts that have timed out
getPDXIdForTypeFailures	Total number of getPDXIdForType operation attempts that have failed
getPDXIdForTypeInProgress	Current number of getPDXIdForType operations being executed
getPDXIdForTypeSuccessful	Total number of getPDXIdForType operations completed successfully
getPDXIdForTypeTime	Total amount of time, in nanoseconds spent doing getPDXIdForType successfully/unsuccessfully.
getPDXIdForTypeTimeouts	Total number of getPDXIdForType operation attempts that have timed out
getPDXTypeByIdFailures	Total number of getPDXTypeById operation attempts that have failed
getPDXTypeByIdInProgress	Current number of getPDXTypeById operations being executed
getPDXTypeByIdSuccessful	Total number of getPDXTypeById operations completed successfully
getPDXTypeByIdTime	Total amount of time, in nanoseconds spent doing getPDXTypeById successfully/unsuccessfully.
getPDXTypeByIdTimeouts	Total number of getPDXTypeById operation attempts that have timed out
getTime	Total amount of time, in nanoseconds spent doing gets
getTimeouts	Total number of get attempts that have timed out
gets	Total number of gets completed successfully
getsInProgress	Current number of gets being executed
invalidateFailures	Total number of invalidate attempts that have failed
invalidateTime	Total amount of time, in nanoseconds spent doing invalidates
invalidateTimeouts	Total number of invalidate attempts that have timed out
invalidates	Total number of invalidates completed successfully
invalidatesInProgress	Current number of invalidates being executed
jtaSynchronizationFailures	Total number of jtaSynchronization attempts that have failed
jtaSynchronizationTime	Total amount of time, in nanoseconds spent doing jtaSynchronizations
jtaSynchronizationTimeouts	Total number of jtaSynchronization attempts that have timed out
jtaSynchronizations	Total number of jtaSynchronizations completed successfully

Statistic	Description
jtaSynchronizationsInProgress	Current number of jtaSynchronizations being executed
keySetFailures	Total number of keySet attempts that have failed
keySetTime	Total amount of time, in nanoseconds spent doing keySets
keySetTimeouts	Total number of keySet attempts that have timed out
keySets	Total number of keySets completed successfully
keySetsInProgress	Current number of keySets being executed
makePrimaryFailures	Total number of makePrimary attempts that have failed
makePrimaryTime	Total amount of time, in nanoseconds spent doing makePrimarys
makePrimaryTimeouts	Total number of makePrimary attempts that have timed out
makePrimarys	Total number of makePrimarys completed successfully
makePrimarysInProgress	Current number of makePrimarys being executed
messageBytesBeingReceived	Current number of bytes consumed by messages being received or processed over a client-to-server connection.
messagesBeingReceived	Current number of message being received off the network or being processed after reception over a client-to-server connection.
pingFailures	Total number of ping attempts that have failed
pingTime	Total amount of time, in nanoseconds spent doing pings
pingTimeouts	Total number of ping attempts that have timed out
pings	Total number of pings completed successfully
pingsInProgress	Current number of pings being executed
primaryAckFailures	Total number of primaryAck attempts that have failed
primaryAckTime	Total amount of time, in nanoseconds spent doing primaryAcks
primaryAckTimeouts	Total number of primaryAck attempts that have timed out
primaryAcks	Total number of primaryAcks completed successfully
primaryAcksInProgress	Current number of primaryAcks being executed
putAllFailures	Total number of putAll attempts that have failed
putAllTime	Total amount of time, in nanoseconds spent doing putAlls
putAllTimeouts	Total number of putAll attempts that have timed out
putAlls	Total number of putAlls completed successfully
putAllsInProgress	Current number of putAlls being executed
putFailures	Total number of put attempts that have failed
putTime	Total amount of time, in nanoseconds spent doing puts
putTimeouts	Total number of put attempts that have timed out
puts	Total number of puts completed successfully
putsInProgress	Current number of puts being executed

Statistic	Description
queryFailures	Total number of query attempts that have failed
queryTime	Total amount of time, in nanoseconds spent doing querys
queryTimeouts	Total number of query attempts that have timed out
querys	Total number of querys completed successfully
querysInProgress	Current number of querys being executed
readyForEvents	Total number of readyForEvents completed successfully
readyForEventsFailures	Total number of readyForEvents attempts that have failed
readyForEventsInProgress	Current number of readyForEvents being executed
readyForEventsTime	Total amount of time, in nanoseconds spent doing readyForEvents
readyForEventsTimeouts	Total number of readyForEvents attempts that have timed out
receivedBytes	Total number of bytes received (as responses) from server over a client-to-server connection.
registerDataSerializers	Total number of registerDataSerializers completed successfully
registerDataSerializersFailures	Total number of registerDataSerializers attempts that have failed
registerDataSerializersInProgress	Current number of registerDataSerializers being executed
registerDataSerializersTime	Total amount of time, in nanoseconds spent doing registerDataSerializers
registerDataSerializersTimeouts	Total number of registerDataSerializers attempts that have timed out
registerInstantiators	Total number of registerInstantiators completed successfully
registerInstantiatorsFailures	Total number of registerInstantiators attempts that have failed
registerInstantiatorsInProgress	Current number of registerInstantiators being executed
registerInstantiatorsTime	Total amount of time, in nanoseconds spent doing registerInstantiators
registerInstantiatorsTimeouts	Total number of registerInstantiators attempts that have timed out
registerInterestFailures	Total number of registerInterest attempts that have failed
registerInterestTime	Total amount of time, in nanoseconds spent doing registerInterests
registerInterestTimeouts	Total number of registerInterest attempts that have timed out
registerInterests	Total number of registerInterests completed successfully
registerInterestsInProgress	Current number of registerInterests being executed
removeAllFailures	Total number of removeAll attempts that have failed
removeAllTime	Total amount of time, in nanoseconds spent doing removeAlls
removeAllTimeouts	Total number of removeAll attempts that have timed out
removeAlls	Total number of removeAlls completed successfully
removeAllsInProgress	Current number of removeAlls being executed
rollbackFailures	Total number of rollback attempts that have failed

Statistic	Description
rollbackTime	Total amount of time, in nanoseconds spent doing rollbacks
rollbackTimeouts	Total number of rollback attempts that have timed out
rollbacks	Total number of rollbacks completed successfully
rollbacksInProgress	Current number of rollbacks being executed
sentBytes	Total number of bytes sent to server over a client-to-server connection.
sizeFailures	Total number of size attempts that have failed
sizeTime	Total amount of time, in nanoseconds spent doing sizes
sizeTimeouts	Total number of size attempts that have timed out
sizes	Total number of sizes completed successfully
sizesInProgress	Current number of sizes being executed
stopCQFailures	Total number of stopCQ attempts that have failed
stopCQTime	Total amount of time, in nanoseconds spent doing stopCQs
stopCQTimeouts	Total number of stopCQ attempts that have timed out
stopCQs	Total number of stopCQs completed successfully
stopCQsInProgress	Current number of stopCQs being executed
txFailoverFailures	Total number of txFailover attempts that have failed
txFailoverTime	Total amount of time, in nanoseconds spent doing txFailovers
txFailoverTimeouts	Total number of txFailover attempts that have timed out
txFailovers	Total number of txFailovers completed successfully
txFailoversInProgress	Current number of txFailovers being executed
unregisterInterestFailures	Total number of unregisterInterest attempts that have failed
unregisterInterestTime	Total amount of time, in nanoseconds spent doing unregisterInterests
unregisterInterestTimeouts	Total number of unregisterInterest attempts that have timed out
unregisterInterests	Total number of unregisterInterests completed successfully
unregisterInterestsInProgress	Current number of unregisterInterests being executed

The primary statistics of ClientSendStats are:

Statistic	Description
addPdxTypeSendFailures	Total number of addPdxType operation's request messages not sent successfully from the client to server
addPdxTypeSendTime	Total amount of time, in nanoseconds spent sending addPdxType operation's request messages successfully/unsuccessfully from the client to server
addPdxTypeSendsInProgress	Current number of addPdxType operation's request messages being send from the client to server
addPdxTypeSendsSuccessful	Total number of addPdxType operation's request messages sent successfully from the client to server

Statistic	Description
authenticationSend	Total number of authentication sends that have completed successfully
authenticationSendFailure	Total number of authentication sends that have failed
authenticationSendInProgress	Current number of authentication sends being executed
authenticationSendTime	Total amount of time, in nanoseconds spent doing authentication sends
clearSendFailures	Total number of clear sends that have failed
clearSendTime	Total amount of time, in nanoseconds spent doing clear sends
clearSends	Total number of clear sends that have completed successfully
clearSendsInProgress	Current number of clear sends being executed
closeCQSendFailures	Total number of closeCQ sends that have failed
closeCQSendTime	Total amount of time, in nanoseconds spent doing closeCQ sends
closeCQSends	Total number of closeCQ sends that have completed successfully
closeCQSendsInProgress	Current number of closeCQ sends being executed
closeConSendFailures	Total number of closeCon sends that have failed
closeConSendTime	Total amount of time, in nanoseconds spent doing closeCon sends
closeConSends	Total number of closeCon sends that have completed successfully
closeConSendsInProgress	Current number of closeCon sends being executed
commitSendFailures	Total number of commit sends that have failed
commitSendTime	Total amount of time, in nanoseconds spent doing commits
commitSends	Total number of commit sends that have completed successfully
commitSendsInProgress	Current number of commit sends being executed
containsKeySendFailures	Total number of containsKey sends that have failed
containsKeySendTime	Total amount of time, in nanoseconds spent doing containsKey sends
containsKeySends	Total number of containsKey sends that have completed successfully
containsKeySendsInProgress	Current number of containsKey sends being executed
createCQSendFailures	Total number of createCQ sends that have failed
createCQSendTime	Total amount of time, in nanoseconds spent doing createCQ sends
createCQSends	Total number of createCQ sends that have completed successfully
createCQSendsInProgress	Current number of createCQ sends being executed
destroyRegionSendFailures	Total number of destroyRegion sends that have failed
destroyRegionSendTime	Total amount of time, in nanoseconds spent doing destroyRegion sends
destroyRegionSends	Total number of destroyRegion sends that have completed successfully
destroyRegionSendsInProgress	Current number of destroyRegion sends being executed

Statistic	Description
destroySendFailures	Total number of destroy sends that have failed
destroySendTime	Total amount of time, in nanoseconds spent doing destroy sends
destroySends	Total number of destroy sends that have completed successfully
destroySendsInProgress	Current number of destroy sends being executed
executeFunctionSendFailures	Total number of Function sends that have failed
executeFunctionSendTime	Total amount of time, in nanoseconds spent doing Function sends
executeFunctionSends	Total number of Function sends that have completed successfully
executeFunctionSendsInProgress	Current number of Function sends being executed
gatewayBatchSendFailures	Total number of gatewayBatch sends that have failed
gatewayBatchSendTime	Total amount of time, in nanoseconds spent doing gatewayBatch sends
gatewayBatchSends	Total number of gatewayBatch sends that have completed successfully
gatewayBatchSendsInProgress	Current number of gatewayBatch sends being executed
getAllSendFailures	Total number of getAll sends that have failed
getAllSendTime	Total amount of time, in nanoseconds spent doing getAll sends
getAllSends	Total number of getAll sends that have completed successfully
getAllSendsInProgress	Current number of getAll sends being executed
getClientPRMetadataSendFailures	Total number of getClientPRMetadata operation's request messages not sent successfully from the client to server
getClientPRMetadataSendTime	Total amount of time, in nanoseconds spent sending getClientPRMetadata operation's request messages successfully/unsuccessfully from the client to server
getClientPRMetadataSendsInProgress	Current number of getClientPRMetadata operation's request messages being send from the client to server
getClientPRMetadataSendsSuccessful	Total number of getClientPRMetadata operation's request messages sent successfully from the client to server
getClientPartitionAttributesSendFailures	Total number of getClientPartitionAttributes operation's request messages not sent successfully from the client to server
getClientPartitionAttributesSendTime	Total amount of time, in nanoseconds spent sending getClientPartitionAttributes operation's request messages successfully/unsuccessfully from the client to server
getClientPartitionAttributesSendsInProgress	Current number of getClientPartitionAttributes operation's request messages being send from the client to server
getClientPartitionAttributesSendsSuccessful	Total number of getClientPartitionAttributes operation's request messages sent successfully from the client to server
getDurableCQsSendFailures	Total number of getDurableCQs sends that have failed
getDurableCQsSendTime	Total amount of time, in nanoseconds spent doing getDurableCQs sends
getDurableCQsSends	Total number of getDurableCQs sends that have completed successfully

Statistic	Description
getDurableCQsSendsInProgress	Current number of getDurableCQs sends being executed
getEntrySendFailures	Total number of getEntry sends that have failed
getEntrySendTime	Total amount of time, in nanoseconds spent sending getEntry messages
getEntrySends	Total number of getEntry sends that have completed successfully
getEntrySendsInProgress	Current number of getEntry sends being executed
getEventValueSend	Total number of getEventValue sends that have completed successfully
getEventValueSendFailure	Total number of getEventValue sends that have failed
getEventValueSendInProgress	Current number of getEventValue sends being executed
getEventValueSendTime	Total amount of time, in nanoseconds spent doing getEventValue sends
getFunctionAttributeSend	Total number of getFunctionAttribute sends that have completed successfully
getFunctionAttributeSendFailure	Total number of getFunctionAttribute sends that have failed
getFunctionAttributeSendInProgress	Current number of getFunctionAttribute sends being executed
getFunctionAttributeSendTime	Total amount of time, in nanoseconds spent doing getFunctionAttribute sends
getPDXIdForTypeSendFailures	Total number of getPDXIdForType operation's request messages not sent successfully from the client to server
getPDXIdForTypeSendTime	Total amount of time, in nanoseconds spent sending getPDXIdForType operation's request messages successfully/unsuccessfully from the client to server
getPDXIdForTypeSendsInProgress	Current number of getPDXIdForType operation's request messages being send from the client to server
getPDXIdForTypeSendsSuccessful	Total number of getPDXIdForType operation's request messages sent successfully from the client to server
getPDXTypeByIdSendFailures	Total number of getPDXTypeById operation's request messages not sent successfully from the client to server
getPDXTypeByIdSendTime	Total amount of time, in nanoseconds spent sending getPDXTypeById operation's request messages successfully/unsuccessfully from the client to server
getPDXTypeByIdSendsInProgress	Current number of getPDXTypeById operation's request messages being send from the client to server
getPDXTypeByIdSendsSuccessful	Total number of getPDXTypeById operation's request messages sent successfully from the client to server
getSendFailures	Total number of get sends that have failed
getSendTime	Total amount of time, in nanoseconds spent doing get sends
getSends	Total number of get sends that have completed successfully
getSendsInProgress	Current number of get sends being executed
invalidateSendFailures	Total number of invalidate sends that have failed
invalidateSendTime	Total amount of time, in nanoseconds spent doing invalidates

Statistic	Description
invalidateSends	Total number of invalidate sends that have completed successfully
invalidateSendsInProgress	Current number of invalidate sends being executed
jtaSynchronizationSendFailures	Total number of jtaSynchronization sends that have failed
jtaSynchronizationSendTime	Total amount of time, in nanoseconds spent doing jtaSynchronizations
jtaSynchronizationSends	Total number of jtaSynchronization sends that have completed successfully
jtaSynchronizationSendsInProgress	Current number of jtaSynchronization sends being executed
keySetSendFailures	Total number of keySet sends that have failed
keySetSendTime	Total amount of time, in nanoseconds spent doing keySet sends
keySetSends	Total number of keySet sends that have completed successfully
keySetSendsInProgress	Current number of keySet sends being executed
makePrimarySendFailures	Total number of makePrimary sends that have failed
makePrimarySendTime	Total amount of time, in nanoseconds spent doing makePrimary sends
makePrimarySends	Total number of makePrimary sends that have completed successfully
makePrimarySendsInProgress	Current number of makePrimary sends being executed
pingSendFailures	Total number of ping sends that have failed
pingSendTime	Total amount of time, in nanoseconds spent doing ping sends
pingSends	Total number of ping sends that have completed successfully
pingSendsInProgress	Current number of ping sends being executed
primaryAckSendFailures	Total number of primaryAck sends that have failed
primaryAckSendTime	Total amount of time, in nanoseconds spent doing primaryAck sends
primaryAckSends	Total number of primaryAck sends that have completed successfully
primaryAckSendsInProgress	Current number of primaryAck sends being executed
putAllSendFailures	Total number of putAll sends that have failed
putAllSendTime	Total amount of time, in nanoseconds spent doing putAll sends
putAllSends	Total number of putAll sends that have completed successfully
putAllSendsInProgress	Current number of putAll sends being executed
putSendFailures	Total number of put sends that have failed
putSendTime	Total amount of time, in nanoseconds spent doing put sends
putSends	Total number of put sends that have completed successfully
putSendsInProgress	Current number of put sends being executed
querySendFailures	Total number of query sends that have failed
querySendTime	Total amount of time, in nanoseconds spent doing query sends

Statistic	Description
querySends	Total number of query sends that have completed successfully
querySendsInProgress	Current number of query sends being executed
readyForEventsSendFailures	Total number of readyForEvents sends that have failed
readyForEventsSendTime	Total amount of time, in nanoseconds spent doing readyForEvents sends
readyForEventsSends	Total number of readyForEvents sends that have completed successfully
readyForEventsSendsInProgress	Current number of readyForEvents sends being executed
registerDataSerializersSendFailures	Total number of registerDataSerializers sends that have failed
registerDataSerializersSendInProgress	Current number of registerDataSerializers sends being executed
registerDataSerializersSendTime	Total amount of time, in nanoseconds spent doing registerDataSerializers sends
registerDataSerializersSends	Total number of registerDataSerializers sends that have completed successfully
registerInstantiatorsSendFailures	Total number of registerInstantiators sends that have failed
registerInstantiatorsSendTime	Total amount of time, in nanoseconds spent doing registerInstantiators sends
registerInstantiatorsSends	Total number of registerInstantiators sends that have completed successfully
registerInstantiatorsSendsInProgress	Current number of registerInstantiators sends being executed
registerInterestSendFailures	Total number of registerInterest sends that have failed
registerInterestSendTime	Total amount of time, in nanoseconds spent doing registerInterest sends
registerInterestSends	Total number of registerInterest sends that have completed successfully
registerInterestSendsInProgress	Current number of registerInterest sends being executed
removeAllSendFailures	Total number of removeAll sends that have failed
removeAllSendTime	Total amount of time, in nanoseconds spent doing removeAll sends
removeAllSends	Total number of removeAll sends that have completed successfully
removeAllSendsInProgress	Current number of removeAll sends being executed
rollbackSendFailures	Total number of rollback sends that have failed
rollbackSendTime	Total amount of time, in nanoseconds spent doing rollbacks
rollbackSends	Total number of rollback sends that have completed successfully
rollbackSendsInProgress	Current number of rollback sends being executed
sizeSendFailures	Total number of size sends that have failed
sizeSendTime	Total amount of time, in nanoseconds spent doing sizes
sizeSends	Total number of size sends that have completed successfully

Statistic	Description
sizeSendsInProgress	Current number of size sends being executed
stopCQSendFailures	Total number of stopCQ sends that have failed
stopCQSendTime	Total amount of time, in nanoseconds spent doing stopCQ sends
stopCQSends	Total number of stopCQ sends that have completed successfully
stopCQSendsInProgress	Current number of stopCQ sends being executed
txFailoverSendFailures	Total number of txFailover sends that have failed
txFailoverSendTime	Total amount of time, in nanoseconds spent doing txFailovers
txFailoverSends	Total number of txFailover sends that have completed successfully
txFailoverSendsInProgress	Current number of txFailover sends being executed
unregisterInterestSendFailures	Total number of unregisterInterest sends that have failed
unregisterInterestSendTime	Total amount of time, in nanoseconds spent doing unregisterInterest sends
unregisterInterestSends	Total number of unregisterInterest sends that have completed successfully
unregisterInterestSendsInProgress	Current number of unregisterInterest sends being executed

Client Connection Pool (PoolStats)

These statistics are in a client and they describe one of the client's connection pools. The primary statistics are:

Statistic	Description
connections	Current number of connections
connectionWaits	Total number of times a thread completed waiting for a connection (by timing out or by getting a connection).
connectionWaitsInProgress	Current number of threads waiting for a connection
connectionWaitTime	Total number of nanoseconds spent waiting for a connection.
connects	Total number of times a connection has been created.
disconnects	Total number of times a connection has been destroyed.
servers	Current number of servers discovered
idleChecks	Total number of checks done for idle expiration.
idleDisconnects	Total number of disconnects done due to idle expiration.
initialContactCount	Number of contacts initially by user
locators	Current number of locators discovered
minPoolSizeConnects	Total number of connects done to maintain minimum pool size.
subscriptionServers	Number of servers hosting this clients subscriptions
locatorRequests	Number of requests from this connection pool to a locator

Statistic	Description
locatorResponses	Number of responses from the locator to this connection pool

Continuous Querying (CqServiceStats)

These statistics are for continuous querying information. The statistics are:

Statistic	Description
numCqsCreated	Number of CQs created.
numCqsActive	Number of CQS actively executing.
numCqsStopped	Number of CQs stopped.
numCqsClosed	Number of CQs closed.
numCqsOnClient	Number of CQs on the client.
numClientsWithCqs	Number of Clients with CQs.
cqQueryExecutionTime	Time taken for CQ Query Execution.
cqQueryExecutionsCompleted	Number of CQ Query Executions.
cqQueryExecutionInProgress	CQ Query Execution In Progress.
numUniqueCqQuery	Number of Unique CQ Querys.
numCqEventsFiltered	Number of CQ events filtered.

Disk Space Usage (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 for oplog files.
maximumSpace	The configured maximum number of bytes allowed in this directory for oplog files. Note that some product configurations allow this maximum to be exceeded.
volumeFreeSpace	The total free space in bytes on the disk volume
volumeFreeSpaceChecks	The total number of disk space checks
volumeFreeSpaceTime	The total time spent checking disk usage
volumeSize	The total size in bytes of the disk volume

Disk Store Statistics (DiskStoreStatistics)

Statistics about a Region's use of the disk. The primary statistics are:

Statistic	Description
backupsCompleted	The number of backups of this disk store that have been taking while this VM was alive.
backupsInProgress	The current number of backups in progress on this disk store.

Statistic	Description
compactableOplogs	Current number of oplogs ready to be compacted.
compactDeletes	Total number of times an oplog compact did a delete.
compactDeleteTime	Total amount of time, in nanoseconds, spent doing deletes during a compact.
compactInserts	Total number of times an oplog compact did a db insert.
compactInsertTime	Total amount of time, in nanoseconds, spent doing inserts during a compact.
compacts	Total number of completed oplog compacts.
compactsInProgress	Current number of oplog compacts that are in progress.
compactTime	Total amount of time, in nanoseconds, spent compacting oplogs.
compactUpdates	Total number of times an oplog compact did an update.
compactUpdateTime	Total amount of time, in nanoseconds, spent doing updates during a compact.
flushedBytes	The total number of bytes written to disk by async queue flushes.
flushes	The total number of times an entry has been flushed from the async queue.
flushesInProgress	Current number of oplog flushes that are in progress.
flushTime	The total amount of time spent doing an async queue flush.
inactiveOplogs	Current number of oplogs that are no longer being written but are not ready to compact.
openOplogs	Current number of oplogs this disk store has open.
oplogReads	Total number of oplog reads.
oplogRecoveries	The total number of oplogs recovered.
oplogRecoveryTime	The total amount of time spent doing an oplog recovery.
oplogRecoveredBytes	The total number of bytes that have been read from oplogs during a recovery.
oplogSeeks	Total number of oplog seeks.
queueSize	The current number of entries in the async queue waiting to be flushed to disk.
readBytes	The total number of bytes that have been read from disk.
reads	The total number of region entries that have been read from disk.
readTime	The total amount of time spent reading from disk.
recoveriesInProgress	Current number of persistent regions being recovered from disk.
recoveredBytes	The total number of bytes that have been read from disk during a recovery.
recoveredEntryCreates	The total number of entry create records processed while recovering oplog data.
recoveredEntryDestroys	The total number of entry destroy records processed while recovering oplog data.
recoveredEntryUpdates	The total number of entry update records processed while recovering oplog data.
recoveredValuesSkipped DueToLRU	The total number of entry values that did not need to be recovered due to the LRU.
recoveryRecordsSkipped	The total number of oplog records skipped during recovery.
recoveryTime	The total amount of time spent doing a recovery.

Statistic	Description
removes	The total number of region entries that have been removed from disk.
removeTime	The total amount of time spent removing from disk.
uncreatedRecoveredRegions	The current number of regions that have been recovered but have not yet been created.
writes	The 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 disk.
writesInProgress	Current number of oplog writes that are in progress.
writeTime	The total amount of time spent writing to disk.
writtenBytes	The total number of bytes that have been written to disk.

Disk Usage and Performance (DiskRegionStatistics)

Statistics regarding operations performed on a disk region for persistence/overflow. The primary statistics are:

Statistic	Description
bytesOnlyOnDisk	The current number bytes on disk and not in memory. This is true of overflowed entries. It is also true of recovered entries that have not yet been faulted in.
entriesInVM	The current number of entries whose value resides in the VM. The value may also have been written to disk.
entriesOnlyOnDisk	The current number of entries whose value is on 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.
readBytes	The total number of bytes that have been read from disk
reads	The total number of region entries that have been read from disk
readTime	The total amount of time spent reading from disk
removes	The total number of region entries that have been removed from disk
removeTime	The total amount of time spent removing from disk
writes	The 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 disk.
writeTime	The total amount of time spent writing to disk
writtenBytes	The total number of bytes that have been written to disk

Distributed System Messaging (DistributionStats)

Statistics on the Tanzu GemFire distribution layer. These statistics can be used to tell how much message traffic exists between this member and other cluster members.

The primary statistics are:

Statistic	Description
asyncConflatedMsgs	The total number of queued conflated messages used for asynchronous queues.
asyncDequeuedMsgs	The total number of queued messages that have been removed from the queue and successfully sent.

Statistic	Description
asyncDistributionTimeoutExceeded	Total number of times the async-distribution-timeout has been exceeded during a socket write.
asyncQueueAddTime	Total amount of time, in nanoseconds, spent in adding messages to async queue.
asyncQueueFlushTime	Total time spent flushing asynchronous queues.
asyncQueueFlushesCompleted	Total number of asynchronous queue flushes completed.
asyncQueueFlushesInProgress	Current number of asynchronous queues being flushed.
asyncQueueRemoveTime	Total amount of time, in nanoseconds, spent in removing messages from async queue.
asyncQueueSize	The current size in bytes used for asynchronous queues.
asyncQueueSizeExceeded	Total number of asynchronous queues that have exceeded max size.
asyncQueueTimeoutExceeded	Total number of asynchronous queues that have timed out by being blocked for more than async-queue-timeout milliseconds.
asyncQueuedMsgs	The total number of queued messages used for asynchronous queues.
asyncQueues	The current number of queues for asynchronous messaging.
asyncSocketWriteBytes	Total number of bytes sent out on non-blocking sockets.
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.
asyncSocketWrites	Total number of non-blocking socket write calls completed.
asyncSocketWritesInProgress	Current number of non-blocking socket write calls in progress.
asyncThreadCompleted	Total number of iterations of work performed by asynchronous message queue threads.
asyncThreadInProgress	Current iterations of work performed by asynchronous message queue threads.
asyncThreadTime	Total time spent by asynchronous message queue threads performing iterations.
asyncThreads	Total number of asynchronous message queue threads.
bufferAcquireTime	Total amount of time, in nanoseconds, spent acquiring a socket
bufferAcquires	Total number of times a buffer has been acquired.
bufferAcquiresInProgress	Current number of threads waiting to acquire a buffer

Statistic	Description
commitMessages	The number of transaction commit messages that this GemFire system has created to be sent. Note that it is possible for a commit to only create one message even though it will end up being sent to multiple recipients.
commitWaits	The number of transaction commits that had to wait for a response before they could complete.
deserializationTime	Total amount of time, in nanoseconds, spent deserializing objects. This includes deserialization that results in a PdxInstance.
deserializations	Total number of object deserialization calls. This includes deserialization that results in a PdxInstance.
deserializedBytes	Total number of bytes read by object deserialization. This includes deserialization that results in a PdxInstance.
elders	Current number of system elders hosted in this member.
failedAccepts	Total number of times an accept (receiver creation) of a connect from some other member has failed
failedConnects	Total number of times a connect (sender creation) to some other member has failed.
finalCheckRequestsReceived	Final check requests that this member has received.
finalCheckRequestsSent	Final check requests that this member has sent.
finalCheckResponsesReceived	Final check responses that this member has received.
finalCheckResponsesSent	Final check responses that this member has sent.
functionExecutionQueueSize	The number of high priority distribution messages currently waiting to be processed.
functionExecutionQueueThrottleCount	The total number of times a thread was delayed in adding a normal message to the high priority queue.
functionExecutionQueueThrottleTime	The total amount of time, in nanoseconds, spent delayed by the high priority queue throttle.
functionExecutionThreadJobs	The number of messages currently being processed by function execution threads
functionExecutionThreadStarts	Total number of times a thread has been created for the pool handling function execution messages.
functionExecutionThreads	The current number of threads dedicated to processing function execution messages. These threads may be idle.
giiSenderQueueSize	The number of gii messages currently waiting for some other resource before they can be processed.
giiSenderThreadJobs	The number of messages currently being processed by GII sender pool processor threads.
giiSenderThreadStarts	Total number of times a thread has been started for the GII sender pool.
giiSenderThreads	The current number of threads dedicated to processing GII sender messages. These threads may be idle.
heartbeatRequestsReceived	Heartbeat request messages that this member has received.

Statistic	Description
heartbeatRequestsSent	Heartbeat request messages that this member has sent.
heartbeatsReceived	Heartbeat messages that this member has received.
heartbeatsSent	Heartbeat messages that this member has sent.
highPriorityQueueSize	The number of high priority distribution messages currently waiting to be processed.
highPriorityQueueThrottleCount	The total number of times a thread was delayed in adding a normal message to the high priority queue.
highPriorityQueueThrottleTime	The total amount of time, in nanoseconds, spent delayed by the high priority queue throttle.
highPriorityThreadJobs	The number of messages currently being processed by high priority processor threads.
highPriorityThreadStarts	Total number of times a thread has been started for the pool handling high priority messages.
highPriorityThreads	The current number of threads dedicated to processing high priority messages. These threads may be idle.
initialImageMessagesInFlight	The number of messages with initial image data sent from this member that have not yet been acknowledged.
initialImageRequestsInProgress	The number of initial images this member is currently receiving.
membershipClientThreadDelayDuration	Total time in milliseconds the membership client thread is delayed for.
membershipClientThreadPauses	Total number of membership client thread pauses (which may or may not be full GC pauses) detected. A membership client thread pause is defined as a system event which kept the membership client thread from completing its work for 3000 or more milliseconds. This threshold can be customized by setting the system property gemfire.statSamplerDelayThreshold (units are milliseconds).
membershipServerThreadDelayDuration	Total time in milliseconds the membership server thread is delayed for.
membershipServerThreadPauses	Total number of membership server thread pauses (which may or may not be full GC pauses) detected. A membership server thread pause is defined as a system event which kept the membership server thread from completing its work for 3000 or more milliseconds. This threshold can be customized by setting the system property gemfire.statSamplerDelayThreshold (units are milliseconds).
messageBytesBeingReceived	Current number of bytes consumed by messages being received or processed.
messageChunksSentSharedOrdered	Total number of message chunks sent on shared ordered senders. Note that a single sentMessage can cause this statistic to be incremented multiple times if that single message is sent to multiple members of the cluster or if the single message is large enough to be split into more than one chunk.
messageChunksSentSharedUnordered	Total number of message chunks sent on shared unordered senders. Note that a single sentMessage can cause this statistic to be incremented multiple times if that single message is sent to multiple members of the cluster or if the single message is large enough to be split into more than one chunk.
messageChunksSentUnshared	Total number of message chunks sent on unshared senders. Note that a single sentMessage can cause this statistic to be incremented multiple times if that single message is sent to multiple members of the cluster or if the single message is large enough to be split into more than one chunk.

Statistic	Description
messageLossEvents	The number of times that message loss was detected
messageSendDelays	The total number of times sending a message was delayed because it needed to reacquire a connection.
messagesBeingReceived	Current number of message being received off the network or being processed after reception.
messagesLost	The total number of messages lost
messagesResent	The number of distribution messages that this GemFire system has resent following message loss.
nodes	The current number of nodes in this distributed system.
overflowQueueSize	The number of normal distribution messages currently waiting to be processed.
overflowQueueThrottleCount	The total number of times a thread was delayed in adding a normal message to the overflow queue.
overflowQueueThrottleTime	The total amount of time, in nanoseconds, spent delayed by the overflow queue throttle.
p2pSocketTimeouts	The total number of socket timeouts on P2P connections. These timeouts occur while creating a new connection and the handshake exceeding a timeout limit.
partitionedRegionQueueSize	The number of high priority distribution messages currently waiting to be processed.
partitionedRegionQueueThrottleCount	The total number of times a thread was delayed in adding a normal message to the high priority queue.
partitionedRegionQueueThrottleTime	The total amount of time, in nanoseconds, spent delayed by the high priority queue throttle.
partitionedRegionThreadJobs	The number of messages currently being processed by partitioned region threads
partitionedRegionThreadStarts	Total number of times a thread has been created for the pool handling partitioned region messages.
partitionedRegionThreads	The current number of threads dedicated to processing partitioned region messages. These threads may be idle.
pdxDeserializations	Total number of pdx deserializations.
pdxDeserializedBytes	Total number of bytes read by pdx deserialization.
pdxInstanceCreations	Total number of times a deserialization created a PdxInstance.
pdxInstanceDeserializationTime	Total amount of time, in nanoseconds, spent deserializing PdxInstances by calling getObject.
pdxInstanceDeserializations	Total number of times getObject has been called on a PdxInstance.
pdxSerializations	Total number of pdx serializations.

Statistic	Description
pdxSerializedBytes	Total number of bytes produced by pdx serialization.
processedMessages	The number of distribution messages that this GemFire system has processed.
processedMessagesTime	The amount of time this distribution manager has spent in message.process().
processingThreadJobs	The number of messages currently being processed by pooled message processor threads.
processingThreadStarts	Total number of times a thread has been started for the pool processing normal messages.
processingThreads	The current number of threads dedicated to processing normal messages. These threads may be idle.
receivedBytes	The number of distribution message bytes that this GemFire system has received.
receivedMessages	The number of distribution messages that this GemFire system has received.
receiverCloses	Total number of receiver connections closed.
receiverCreates	Total number of receiver connections created.
receiverDirectBufferSize	Current number of bytes allocated from direct memory as buffers for incoming messages. Note: these buffers can be used by receivers or senders. Senders use them for receiving direct replies and the handshake response.
receiverHeapBufferSize	Current number of bytes allocated from Java heap memory as buffers for incoming messages. Note: these buffers can be used by receivers or senders. Senders use them for receiving direct replies and the handshake response.
receiverSocketConnects	Total number of receiver socket connects.
receivers	Deprecated, use receiversShared instead.
receiversShared	Current number of sockets/threads dedicated to receiving messages from shared senders in other members.
receiversTO	Deprecated, use receiversUnshared instead.
receiversUnshared	Current number of sockets/threads dedicated to receiving messages from unshared senders in other members.
reconnectAttempts	Deprecated: use reconnects instead.
reconnects	Total number of times a sender connection was reconnected after an unexpected disconnect.
replyHandoffTime	Total number of seconds to switch thread contexts from processing thread to application thread.
replyTimeouts	Total number of message replies that have timed out.
replyWaitTime	Total time spent waiting for a reply to a message.
replyWaitsCompleted	Total number of times waits for a reply have completed.
replyWaitsInProgress	Current number of threads waiting for a reply.

Statistic	Description
senderAsyncCreateRequests	The total number of asynchronous sender create requests.
senderCloses	Total number of sender connections closed.
senderCreateTime	Total amount of time, in nanoseconds, spent creating a sender.
senderCreates	Total number of sender connections created.
senderCreatesInProgress	Current number of sender creations in progress.
senderDirectBufferSize	Current number of bytes allocated from direct memory as buffers for outgoing messages. Note: these buffers can be used by receivers or senders. Receivers use them for sending direct replies and the handshake response.
senderHeapBufferSize	Current number of bytes allocated from Java heap memory as buffers for outgoing messages. Note: these buffers can be used by receivers or senders. Receivers use them for sending direct replies and the handshake response.
senderSocketConnects	Total number of sender socket connects.
senderTimeouts	Total number of times a sender socket has been closed because of the socket-lease-time gemfire property.
sendersPurgedFromPool	The total number of senders removed from the pool because they had already timed out.
sendersSO	Current number of shared sockets dedicated to sending ordered messages. Shared sockets are a limited resource that multiple threads may attempt to use and only one thread can use it at a time. conserve-sockets=true causes all senders to be shared ones.
sendersSU	Current number of shared sockets dedicated to sending unordered messages. Shared sockets are a limited resource that multiple threads may attempt to use and only one thread can use it at a time. conserve-sockets=true causes all senders to be shared ones.
sendersTO	Deprecated, use unsharedSenders instead.
sendersUnshared	Current number of unshared sockets dedicated to sending ordered messages. Unshared senders can be pooled or owned by a single thread. When owned, the owner thread has exclusive access to it and no other thread will attempt to use it. Unshared senders will only exist if conserve-sockets is set to false.
sendersUnsharedDisconnectChecks	Total number of times an unshared sender has been checked for a disconnect.
sendersUnsharedThreadOwned	Current number of unshared sockets dedicated to sending ordered messages that are currently not in use but owned by the thread that last sent a message on them. This happens with certain messages that can not tell if it is safe to return the sender to the pool.
sentBytes	The number of distribution message bytes that this GemFire system has sent.
sentDirectReplyMessages	The total number of sentUnsharedMessages that requested a direct reply.
sentMessageQueueSize	The total number of distribution messages that are queued in all sent message queues.
sentMessagesAcked	The total number of distribution messages that have been acknowledged by the receiver.
sentMessagesTime	The total amount of time this distribution manager has spent sending messages. This includes broadcastMessagesTime.

Statistic	Description
sentSharedOrderedMessages	The total number of sentMessages sent on shared ordered senders.
sentSharedOrderedMessagesInProgress	The current number of threads sending a message on a shared ordered sender.
sentSharedOrderedMessagesTime	The total amount of time, in nanoseconds, spent sending shared ordered messages.
sentSharedUnorderedMessages	The total number of sentMessages sent on shared unordered senders.
sentSharedUnorderedMessagesInProgress	The current number of threads sending a message on a shared unordred sender.
sentSharedUnorderedMessagesTime	The total amount of time, in nanoseconds, spent sending shared unordered messages.
sentUnsharedMessages	The total number of sentMessages sent on unshared senders.
sentUnsharedMessagesInProgress	The current number of threads sending a message on an unshared sender.
sentUnsharedMessagesTime	The total amount of time, in nanoseconds, spent sending unshared messages.
sentUnsharedThreadOwnedMessages	The total number of sentUnsharedMessages that required the sender to be thread owned.
serialPooledThread	The number of threads created in the SerialQueuedExecutorPool.
serialPooledThreadJobs	The number of messages currently being processed by pooled serial processor threads.
serialPooledThreadStarts	Total number of times a thread has been created for the serial pool(s).
serialQueueBytes	The approximate number of bytes consumed by serial distribution messages currently waiting to be processed.
serialQueueSize	The number of serial distribution messages currently waiting to be processed.
serialQueueThrottleCount	The total number of times a thread was delayed in adding a ordered message to the serial queue.
serialQueueThrottleTime	The total amount of time, in nanoseconds, spent delayed by the serial queue throttle.
serialThreadJobs	The number of messages currently being processed by serial threads.
serialThreadStarts	Total number of times a thread has been started for the serial message executor.
serialThreads	The current number of threads dedicated to processing serial/ordered messages. These threads may be idle.
serializationTime	Total amount of time, in nanoseconds, spent serializing objects. This includes pdx serializations.
serializations	Total number of object serialization calls. This includes pdx serializations.

Statistic	Description
serializedBytes	Total number of bytes produced by object serialization. This includes pdx serializations.
suspectsReceived	Suspect member messages that this member has received.
suspectsSent	Suspect member messages that this member has sent.
syncSocketWriteBytes	Total number of bytes sent out in synchronous/blocking mode on sockets.
syncSocketWriteTime	Total amount of time, in nanoseconds, spent in synchronous/blocking socket write calls.
syncSocketWrites	Total number of completed synchronous/blocking socket write calls.
syncSocketWritesInProgress	Current number of synchronous/blocking socket write calls in progress.
tcpAcquireConnectionPending	Current number of connections pending acquisition in the membership TCP messenger
tcpChannelsActivated	Total number of netty channels activated by the TCP Messenger
tcpChannelsDeactivated	Total number of netty channels deactivated which where created by the TCP messenger
tcpChannelsRegistered	Total number of netty channels registered which where created by the TCP messenger
tcpChannelsUnregistered	Total number of netty channels unregistered which where created by the TCP messenger
tcpConnectionGracefulCloses	Total number of graceful termination of connections established by the membership TCP messenger
tcpConnectionUnexpectedCloses	Total number of unexpected connection termination in the membership TCP messenger
tcpDispatchRequestTime	The total amount of time spent deserializing and dispatching TCP messages in the tcp messenger thread.
tcpFinalCheckRequestsReceived	TCP final check requests that this member has received.
tcpFinalCheckRequestsSent	TCP final check requests that this member has sent.
tcpFinalCheckResponsesReceived	TCP final check responses that this member has received.
tcpFinalCheckResponsesSent	TCP final check responses that this member has sent.
tcpMessagesReceived	Total number of application messages received by the TCP messenger
tcpMessagesSent	Total number of application messages sent by the TCP messenger
waitingQueueSize	The number of distribution messages currently waiting for some other resource before they can be processed.

Statistic	Description
waitingThreadJobs	The number of messages currently being processed by waiting pool processor threads.
waitingThreadStarts	Total number of times a thread has been started for the waiting pool.
waitingThreads	The current number of threads dedicated to processing messages that may wait for a resource. These threads may be idle.

Distribution Statistics Related to Slow Receivers

The distribution statistics provide statistics pertaining to slow receivers. The primary statistics are:

Statistic	Description
asyncDistributionTimeoutExceeded	Incremented every time an asyncSocketWrite has exceeded async-distribution-timeout and an async queue has been created.
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.
asyncQueueSizeExceeded	Incremented every time a queue has exceeded async-max-queue-size and the receiver has been sent a disconnect message.
asyncQueueTimeoutExceeded	Incremented every time a queue flushing has exceeded async-queue-timeout and the receiver has been sent a disconnect message.
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.

Distributed Lock Services (DLockStats)

These statistics are for distributed lock services. The primary statistics are:

Statistic	Description
becomeGrantorRequests	Total number of times this member has explicitly requested to become lock grantor.
createGrantorsCompleted	Total number of initial grantors created in this process.
createGrantorsInProgress	Current number of initial grantors being created in this process.
destroyReads	The current number of DLockService destroy read locks held by this process.
destroyReadWaitFailedTime	Total time spent waiting for a DLockService destroy read lock that was not obtained.
destroyReadWaitsCompleted	Total number of times a DLockService destroy read lock wait has completed successfully.
destroyReadWaitsFailed	Total number of times a DLockService destroy read lock wait has completed unsuccessfully.
destroyReadWaitsInProgress	Current number of threads waiting for a DLockService destroy read lock.
destroyReadWaitTime	Total time spent waiting for a DLockService destroy read lock that was obtained.
destroyWrites	The current number of DLockService destroy write locks held by this process.
destroyWriteWaitFailedTime	Total time spent waiting for a DLockService destroy write lock that was not obtained.
destroyWriteWaitsCompleted	Total number of times a DLockService destroy write lock wait has completed successfully.

Statistic	Description
destroyWriteWaitsFailed	Total number of times a DLockService destroy write lock wait has completed unsuccessfully.
destroyWriteWaitsInProgress	Current number of thwrites waiting for a DLockService destroy write lock.
destroyWriteWaitTime	Total time spent waiting for a DLockService destroy write lock that was obtained.
grantors	The current number of lock grantors hosted by this system member.
grantorThreadExpireAndGrantLocksTime	Total time spent by grantor thread(s) performing expireAndGrantLocks tasks.
grantorThreadHandleRequestTimeoutsTime	Total time spent by grantor thread(s) performing handleRequestTimeouts tasks.
grantorThreadRemoveUnusedTokensTime	Total time spent by grantor thread(s) performing removeUnusedTokens tasks.
grantorThreadsCompleted	Total number of iterations of work performed by grantor thread(s).
grantorThreadsInProgress	Current iterations of work performed by grantor thread(s).
grantorThreadTime	Total time spent by grantor thread(s) performing all grantor tasks.
grantorWaitFailedTime	Total time spent waiting for the grantor latch which resulted in failure.
grantorWaitsCompleted	Total number of times waiting threads completed waiting for the grantor latch to open.
grantorWaitsFailed	Total number of times waiting threads failed to finish waiting for the grantor latch to open.
grantorWaitsInProgress	Current number of threads waiting for grantor latch to open.
grantorWaitTime	Total time spent waiting for the grantor latch which resulted in success.
grantWaitDestroyedTime	Total time spent granting of lock requests that failed because lock service was destroyed.
grantWaitFailedTime	Total time spent granting of lock requests that failed because try locks failed.
grantWaitNotGrantorTime	Total time spent granting of lock requests that failed because not grantor.
grantWaitNotHolderTime	Total time spent granting of lock requests that failed because reentrant was not holder.
grantWaitsCompleted	Total number of times granting of a lock request has completed by successfully granting the lock.
grantWaitsDestroyed	Total number of times granting of lock request failed because lock service was destroyed.
grantWaitsFailed	Total number of times granting of lock request failed because try locks failed.
grantWaitsInProgress	Current number of distributed lock requests being granted.
grantWaitsNotGrantor	Total number of times granting of lock request failed because not grantor.
grantWaitsNotHolder	Total number of times granting of lock request failed because reentrant was not holder.
grantWaitsSuspended	Total number of times granting of lock request failed because lock service was suspended.
grantWaitsTimeout	Total number of times granting of lock request failed because of timeout.
grantWaitSuspendedTime	Total time spent granting of lock requests that failed because lock service was suspended.
grantWaitTime	Total time spent attempting to grant a distributed lock.
grantWaitTimeoutTime	Total time spent granting of lock requests that failed because of timeout.
lockReleasesCompleted	Total number of times distributed lock release has completed.
lockReleasesInProgress	Current number of threads releasing a distributed lock.

Statistic	Description
lockReleaseTime	Total time spent releasing a distributed lock.
lockWaitFailedTime	Total time spent waiting for a distributed lock that we failed to obtain.
lockWaitsCompleted	Total number of times distributed lock wait has completed by successfully obtained the lock.
lockWaitsFailed	Total number of times distributed lock wait has completed by failing to obtain the lock.
lockWaitsInProgress	Current number of threads waiting for a distributed lock.
lockWaitTime	Total time spent waiting for a distributed lock that was obtained.
pendingRequests	The current number of pending lock requests queued by grantors in this process.
requestQueues	The current number of lock request queues used by this system member.
serialQueueSize	The number of serial distribution messages currently waiting to be processed.
serialThreads	The number of threads currently processing serial/ordered messages.
serviceCreateLatchTime	Total time spent creating lock services before releasing create latches.
serviceCreatesCompleted	Total number of lock services created in this process.
serviceCreatesInProgress	Current number of lock services being created in this process.
serviceInitLatchTime	Total time spent creating lock services before releasing init latches.
services	The current number of lock services used by this system member.
createGrantorTime	Total time spent waiting create the initial grantor for lock services.
tokens	The current number of lock tokens used by this system member.
waitingQueueSize	The number of distribution messages currently waiting for some other resource before they can be processed.
waitingThreads	The number of threads currently processing messages that had to wait for a resource.

Function Execution (FunctionStatistics)

These are the statistics for each execution of the function. The primary statistics are:

Statistic	Description
functionExecutionCalls	Total number of FunctionService.execute() calls for given function
functionExecutionsCompleted	Total number of completed function.execute() calls for given function
functionExecutionsCompletedProcessingTime	Total time consumed for all completed invocations of the given function
functionExecutionsExceptions	Total number of Exceptions Occurred while executing 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.
functionExecutionsRunning	number of currently running invocations of the given function
resultsReceived	Total number of results received and passed to the ResultCollector
resultsSentToResultCollector	Total number of results sent to the ResultCollector

Gateway Queue (GatewaySenderStatistics)

These statistics are for outgoing gateway queue and its connection. The primary statistics are:

Statistic	Description
batchDistributionTime	time spent distributing batches of events to receivers
batchesDistributed	batches of events sent to receivers
batchesRedistributed	batches of events resent to receivers
batchesResized	batches of events resized because they were too large
batchesWithIncompleteTransactions	batches of events sent to receivers with incomplete transactions
conflationIndexesSize	entries in the conflation indexes map
deltaEventsCreated	delta events created
deltaEventsDistributed	delta events sent to the receiver
deltaEventsQueued	delta events added to the queue
deltaExceptionsReceived	exceptions received from the receiver meaning the full value was not available
enqueueFullValueFailedEntryDestroyed	enqueue full value requests that failed due to the region entry being destroyed
eventQueueSize	size of the queue
eventQueueTime	time spent queueing event
eventsConflatedFromBatches	events conflated from batches
eventsDistributed	events sent to receivers
eventsDroppedDueToPrimarySenderNotRunning	events dropped because the primary sender is not running
eventsExceedingAlertThreshold	events in the queue longer than the alert threshold
eventsFiltered	events filtered by a GatewayEventFilter
eventsNotQueuedConflated	events received but not added to the queue because it already contained an event with the same key
eventsProcessedByPQRM	events processed by a parallel queue removal message
eventsQueued	events added to the queue
eventsReceived	events received by the sender
failedWanCopyRegionTime	total time spent executing failed wan-copy region command
fullValueEventsCreated	full value events created after the remote site was unable to process a delta event
fullValueEventsCreatedWithDeltaAvailable	full value events created even though the delta was available
fullValueEventsDistributed	full value events sent to the receiver
gatewayEventFailureListenerCallsCompleted	total number of gateway event failure listener calls has completed
gatewayEventFailureListenerCallsInProgress	current number of gateway event failure listener calls in progress
gatewayEventFailureListenerCallsTime	total time spent doing gateway event failure listener calls

Statistic	Description
loadBalanceTime	time spent load balancing the sender's connections to the receivers
loadBalancesCompleted	receiver connection load balances completed
loadBalancesInProgress	receiver connection load balances in progress
notQueuedEvent	events not added to the queue
regionsCopied	total number of regions copied by the wan-copy region command
regionsFailedToBeCopied	total number of regions that failed to be copied by the wan-copy region command
secondaryEventQueueSize	size of the secondary queue
synchronizationEventsEnqueued	synchronization events added to the queue
synchronizationEventsProvided	synchronization events provided to other members
tempQueueSize	size of the temporary events queue
unprocessedEventMapSize	entries in the secondary sender's unprocessed event map
unprocessedEventsAddedBySecondary	events added to the secondary sender's unprocessed event map by the secondary
unprocessedEventsRemovedByPrimary	events removed from the secondary sender's unprocessed event map by the primary
unprocessedEventsRemovedByTimeout	events removed from the secondary sender's unprocessed event map by a timeout
unprocessedTokenMapSize	entries in the secondary sender's unprocessed token map
unprocessedTokensAddedByPrimary	tokens added to the secondary sender's unprocessed token map by the primary
unprocessedTokensRemovedBySecondary	tokens removed from the secondary sender's unprocessed token map by the secondary
unprocessedTokensRemovedByTimeout	tokens removed from the secondary sender's unprocessed token map by a timeout
wanCopyRegionFilterEventsTime	total time spent filtering events during wan-copy region
wanCopyRegionInProgress	current number of wan-copy region command in progress
wanCopyRegionTime	total time spent executing wan-copy region command

Indexes (IndexStats)

Query-Independent Statistics on Indexes

Statistic	Description
numKeys	Number of keys in this index
numUpdates	Number of updates that have completed on this index
numValues	Number of values in this index
updatesInProgress	Current number of updates in progress.
updateTime	Total time spent updating this index
numUses	Number of times this index has been used while executing a queries
usesInProgress	Current number of uses in progress.
useTime	Total time spent using this index

Query-Dependent Statistics on Indexes

Statistic	Description
numUses	Number of times the Index has been used 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.
useTime	Total time during the use of the Index for querying.

JVM Performance

Tanzu GemFire JVM Resource Manager (ResourceManagerStats)

Statistics related to the Tanzu GemFire's resource manager. Use these to help analyze and tune your JVM memory settings and the Tanzu GemFire resource manager settings. The primary statistics are:

Statistic	Description
autoRebalanceAttempts	Total number of cache auto-rebalance attempts.
criticalThreshold	The currently set heap critical threshold value in bytes
entryExpirationTime	Total amount of time, in nanoseconds, spent doing entry expirations.
entryExpirations	Total number of entry expirations that have completed.
entryExpirationsInProgress	Current number of entry expirations in progress.
entryExpireDestroys	Total number of destroys when expiring the entry.
entryExpireInvalidates	Total number of invalidates when expiring the entry.
entryExpireLocalDestroys	Total number of local destroys when expiring the entry.
entryExpireLocalInvalidates	Total number of local invalidates when expiring the entry.
evictionStartEvents	Total number of times heap usage went over eviction threshold.
evictionStopEvents	Total number of times heap usage fell below eviction threshold.
evictionThreshold	The currently set heap eviction threshold value in bytes
heapCriticalEvents	Total number of times the heap usage went over critical threshold.
heapSafeEvents	Total number of times the heap usage fell below critical threshold.
latestLastAccessTimeChecks	Total number of LatestLastAccessTime checks on other members.
longRunningThreads	Number of monitored threads that have not completed their task within the thread-monitor-time-limit-ms interval.
numThreadsStuck	Deprecated. Use longRunningThreads instead.
offHeapCriticalEvents	Total number of times off-heap usage went over critical threshold.
offHeapCriticalThreshold	The currently set off-heap critical threshold value in bytes
offHeapEvictionStartEvents	Total number of times off-heap usage went over eviction threshold.
offHeapEvictionStopEvents	Total number of times off-heap usage fell below eviction threshold.

Statistic	Description
offHeapEvictionThreshold	The currently set off-heap eviction threshold value in bytes
offHeapSafeEvents	Total number of times off-heap usage fell below critical threshold.
rebalanceBucketCreateBytes	Total bytes created while directing bucket create operations for rebalancing.
rebalanceBucketCreateTime	Total time spent directing bucket create operations for rebalancing.
rebalanceBucketCreatesCompleted	Total number of bucket create operations directed for rebalancing.
rebalanceBucketCreatesFailed	Total number of bucket create operations directed for rebalancing that failed.
rebalanceBucketCreatesInProgress	Current number of bucket create operations being directed for rebalancing.
rebalanceBucketRemovesBytes	Total bytes removed while directing bucket remove operations for rebalancing.
rebalanceBucketRemovesCompleted	Total number of bucket remove operations directed for rebalancing.
rebalanceBucketRemovesFailed	Total number of bucket remove operations directed for rebalancing that failed.
rebalanceBucketRemovesInProgress	Current number of bucket remove operations being directed for rebalancing.
rebalanceBucketRemovesTime	Total time spent directing bucket remove operations for rebalancing.
rebalanceBucketTransfersBytes	Total bytes transfered while directing bucket transfer operations for rebalancing.
rebalanceBucketTransfersCompleted	Total number of bucket transfer operations directed for rebalancing.
rebalanceBucketTransfersFailed	Total number of bucket transfer operations directed for rebalancing that failed.
rebalanceBucketTransfersInProgress	Current number of bucket transfer operations being directed for rebalancing.
rebalanceBucketTransfersTime	Total time spent directing bucket transfer operations for rebalancing.
rebalanceMembershipChanges	The number of times that membership has changed during a rebalance
rebalancePrimaryTransferTime	Total time spent directing primary transfer operations for rebalancing.
rebalancePrimaryTransfersCompleted	Total number of primary transfer operations directed for rebalancing.
rebalancePrimaryTransfersFailed	Total number of primary transfer operations directed for rebalancing that failed.
rebalancePrimaryTransfersInProgress	Current number of primary transfer operations being directed for rebalancing.
rebalanceTime	Total time spent directing cache rebalance operations.
rebalancesCompleted	Total number of cache rebalance operations directed by this process.
rebalancesInProgress	Current number of cache rebalance operations being directed by this process.
resourceEventQueueSize	Pending events for thresholdEventProcessor thread
resourceEventsDelivered	Total number of resource events delivered to listeners
restoreRedundanciesCompleted	Total number of cache restore redundancy operations directed by this process.
restoreRedundanciesInProgress	Current number of cache restore redundancy operations being directed by this process.

Statistic	Description
restoreRedundancyTime	Total time spent directing cache restore redundancy operations.
tenuredHeapUsed	Total memory used in the tenured/old space
threadMonitorOverheadTime	Total amount of time, in nanoseconds, the thread monitor has spent gathering thread information.
thresholdEventProcessorThreadJobs	Number of jobs currently being processed by the thresholdEventProcessorThread

JVM Java Runtime (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 java VM on its machine.
daemonThreads	Current number of live daemon threads in this VM.
fdLimit	Maximum number of file descriptors
fdsOpen	Current number of open file descriptors
freeMemory	An approximation of the total amount of memory currently available for future allocated objects, measured in bytes.
loadedClasses	Total number of classes loaded since vm started.
maxMemory	The maximum amount of memory that the VM will attempt to use, measured in bytes.
peakThreads	High water mark of live threads in this VM.
pendingFinalization	Number of objects that are pending finalization in the java VM.
processCpuTime	CPU time used by the process in nanoseconds.
threads	Current number of live threads (both daemon and non-daemon) in this VM.
threadStarts	Total number of times a thread has been started since this vm started.
totalMemory	The total amount of memory currently available for current and future objects, measured in bytes.
unloadedClasses	Total number of classes unloaded since vm started.

JVM Garbage Collection (VMGCStats)

These statistics show how much time used by different JVM garbage collection. 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.

JVM Garbage Collector Memory Pools (VMMemoryPoolStats)

These statistics describe memory usage in different 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 for this pool in bytes
collectionUsedMemory	The estimated amount of used memory after that last garbage collection of this pool, measured in bytes.
currentCommittedMemory	The amount of committed memory for this pool, measured in bytes.
currentInitMemory	Initial memory the vm requested from the operating system for this pool
currentMaxMemory	The maximum amount of memory this pool can have in bytes.
currentUsedMemory	The estimated amount of used memory currently in use for this pool, measured in bytes.
usageExceeded	Total number of times that memory usage in this pool exceeded the usageThreshold
usageThreshold	The usage threshold for this pool in bytes

JVM Heap Memory Usage (VMMemoryUsageStats)

Show details on how the Java heap memory is being used. The primary statistics are:

Statistic	Description
committedMemory	The amount of committed memory for this area, measured in bytes.
initMemory	Initial memory the vm requested from the operating system for this area
maxMemory	The maximum amount of memory this area can have in bytes.
usedMemory	The amount of used memory for this area, measured in bytes.

JVM Thread Statistics (VMThreadStats)

Show details about the JVM thread. The primary statistics are:

Statistic	Description
blocked	Total number of times this thread blocked to enter or reenter a monitor
blockedTime	Total amount of elapsed time, approximately, that this thread has spent blocked to enter or reenter a monitor. May need to be enabled by setting -Dgemfire.enableContentionTime=true
cpuTime	Total cpu time for this thread. May need to be enabled by setting -Dgemfire.enableCpuTime=true.
inNative	1 if the thread is in native code.
lockOwner	The thread id that owns the lock that this thread is blocking on.
suspended	1 if this thread is suspended
userTime	Total user time for this thread. May need to be enabled by setting -Dgemfire.enableCpuTime=true.
waited	Total number of times this thread waited for notification.

Statistic	Description
waitedTime	Total amount of elapsed time, approximately, that this thread has spent waiting for notification. May need to be enabled by setting -Dgemfire.enableContentionTime=true

Locator (LocatorStats)

These statistics are on the Tanzu GemFire locator. The primary statistics are:

Statistic	Description
locatorRequests	Total number of requests this locator has received
locatorResponses	Total number of responses this locator has sent
locators	Number of locators known to this locator
requestProcessingTime	Total time spent processing requests
requestsInProgress	The number of requests currently being processed by the thread pool.
responseProcessingTime	Total time spent sending responses
serverLoadUpdates	Total number of times a server load update has been received.
servers	Number of servers this locator knows about

Lucene Indexes (LuceneIndexStats)

These statistics quantify the use of Lucene indexes. The primary statistics are:

Statistic	Description
queryExecutions	The number of Lucene queries executed on this member.
queryExecutionTime	The amount of time in nanoseconds spent executing Lucene queries.
queryExecutionsInProgress	The number of query executions currently in progress.
queryExecutionTotalHits	The total number of documents returned by query executions.
repositoryQueryExecutions	The number of Lucene repository queries executed on this member.
repositoryQueryExecutionTime	The amount of time in nanoseconds spent executing Lucene repository queries.
repositoryQueryExecutionsInProgress	The number of repository query executions currently in progress.
repositoryQueryExecutionTotalHits	The total number of documents returned by repository query executions.
updates	The number of Lucene index documents added or removed on this member.
updateTime	The amount of time in nanoseconds spent adding or removing documents from the index.
updatesInProgress	The number of index updates in progress.
commits	The number of Lucene index commits on this member.
commitTime	The amount of time in nanoseconds spent in Lucene index commits.
commitsInProgress	The number of Lucene index commits in progress.
documents	The number of documents in the index.

Off-Heap (OffHeapMemoryStats)

These statistics quantify the use of off-heap memory. The primary statistics are:

Statistic	Description
defragmentations	The total number of times off-heap memory has been defragmented.
defragmentationsInProgress	Current number of defragment operations currently in progress.
defragmentationTime	The total time spent defragmenting off-heap memory.
fragmentation	The percentage of off-heap free memory that is fragmented. Updated every time a defragmentation is performed.
fragments	The number of fragments of free off-heap memory. Updated every time a defragmentation is done.
freeMemory	The amount of off-heap memory, in bytes, that is not being used.
largestFragment	The largest fragment of off-heap memory that can be used to allocate an object. Updated every time a defragmentation is done and periodically according to off-heap-stats-update-frequency-ms system property (default 3600 seconds)
maxMemory	The maximum amount of off-heap memory, in bytes. This is the amount of memory allocated at startup and does not change.
objects	The number of objects stored in off-heap memory.
reads	The total number of reads of off-heap memory. Only reads of a full object increment this statistic. If only a part of the object is read this statistic is not incremented.
usedMemory	The amount of off-heap memory, in bytes, that is being used to store data.

Operating System Statistics - Linux

Operating system statistics are available only for Linux operating systems.

Linux Process Performance (ProcessStats)

Operating system statistics on the member's process. The statistics are:

Statistic	Description
bytesRead	The number of bytes read by the process. This includes all I/O activity generated by the process to include file, network, and device I/Os
bytesWritten	The number of bytes written by the process. This includes all I/O activity generated by the process to include file, network, and device I/Os
contextSwitches	A snapshot of the context switches the process has done. Since the context switches could be voluntary and non-voluntary, this gives the sum of both.
cpuLoad	CPU usage of this process.
cpuLoadCumulative	Gets the cumulative CPU usage of this process.
kernelTime	Gets kernel time used by the process.
majorFaults	Gets the number of major (hard) faults the process has made which have required loading a memory page from disk.
minorFaults	Gets the number of minor (soft) faults the process has made which have not required loading a memory page from disk. Sometimes called reclaims.

Statistic	Description
openFiles	Gets the number of open file handles (or network connections) that belong to the process.
residentSetSize	Gets the Resident Set Size (RSS). Used to show how much memory is allocated to that process and is in RAM. It does not include memory that is swapped out. It does include memory from shared libraries as long as the pages from those libraries are actually in memory. It does include all stack and heap memory.
threadCount	Gets the number of threads being executed by this process.
userTime	Gets user time used by the process.
virtualSize	Gets the Virtual Memory Size (VSZ). Includes all memory that the process can access, including memory that is swapped out and memory that is from shared libraries.

Linux Operating System (OperatingSystemStats)

Operating system statistics on the member's machine. These can be used to determine total cpu, memory, and disk usage on the machine. The statistics are:

Statistic	Description
contextSwitches	Get the number of system-wide context switches which have occurred.
cpuActive	The percentage of the total available time that has been used in a non-idle state.
cpuIdle	The percentage of the total available time that has been spent sleeping.
cpuNice	The percentage of the total available time that has been used to execute user code in processes with low priority.
cpuSteal	Steal time is the amount of time the operating system wanted to execute, but was not allowed to by the hypervisor.
cpuSystem	The percentage of the total available time that has been used to execute system (i.e. kernel) code.
cpuUser	The percentage of the total available time that has been used to execute user code.
interrupts	Get the number of system-wide interrupts which have occurred.
iowait	The percentage of the total available time that has been used to wait for I/O to complete.
irq	The percentage of the total available time that has been used servicing interrupts.
logicalProcessorCount	Get the number of logical CPUs available for processing. This value may be higher than physical CPUs if hyperthreading is enabled.
loopbackBytes	The number of network bytes sent (or received) on the loopback interface
loopbackPackets	The number of network packets sent (or received) on the loopback interface
maxFileDescriptors	The maximum number of open file descriptors.
memoryAvailable	The amount of physical memory currently available, in bytes.
memoryPageSize	The number of bytes in a memory page
memoryTotal	The amount of actual physical memory, in bytes.
openFileDescriptors	The current number of open file descriptors

Statistic	Description
physicalProcessorCount	Get the number of physical CPUs/cores available for processing.
processCount	Get the number of processes currently running.
recvBytes	The total number of network bytes received (excluding loopback)
recvDrops	The total number network receives dropped
recvErrors	The total number of network receive errors
recvPackets	The total number of network packets received (excluding loopback)
softirq	The percentage of the total available time that has been used servicing softirqs.
swapPagesIn	Number of pages read from paging/swap file(s) to resolve hard page faults.
swapPagesOut	Number of pages read from paging/swap file(s) to resolve hard page faults.
swapTotal	The current size of the paging/swap file(s), in bytes. If the paging/swap file can be extended, this is a soft limit.
swapUsed	The current memory committed to the paging/swap file(s), in bytes.
systemCpuLoad	Recent CPU load measured between statistic polls
systemCpuLoadTicksIDLE	Time spent in Idle
systemCpuLoadTicksIOWAIT	Time spent in IOWait
systemCpuLoadTicksIRQ	Time spent in IRQ
systemCpuLoadTicksNICE	Time spent in Nice
systemCpuLoadTicksSOFTIRQ	Time spent in SoftIRQ
systemCpuLoadTicksSTEAL	Time spent in Steal
systemCpuLoadTicksSYSTEM	Time spent in System
systemCpuLoadTicksUSER	Time spent in User
systemLoadAverage1	The system load average is the sum of the number of runnable entities queued to the available processors and the number of runnable entities running on the available processors averaged over 1 minute.
systemLoadAverage15	The system load average is the sum of the number of runnable entities queued to the available processors and the number of runnable entities running on the available processors averaged over 15 minutes.
systemLoadAverage5	The system load average is the sum of the number of runnable entities queued to the available processors and the number of runnable entities running on the available processors averaged over 5 minutes.
tasksQueuedTime	sum of the time spent waiting to run by tasks on all CPUs
tasksRunningTime	sum of the time spent running by tasks on all CPUs
tasksTimeslicesScheduled	sum of the number of timeslices given to tasks on all CPUs

Statistic	Description
tcpv4ConnectionFailures	Connections Established is the number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT.
tcpv4ConnectionsActive	Connections Active is the number of times TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state. In other words, it shows a number of connections which are initiated by the local computer. The value is a cumulative total.
tcpv4ConnectionsEstablished	Connection Failures is the number of times TCP connections have made a direct transition to the CLOSED state from the SYN-SENT state or the SYN-RCVD state, plus the number of times TCP connections have made a direct transition to the LISTEN state from the SYN-RCVD state.
tcpv4ConnectionsPassive	Connections Passive is the number of times TCP connections have made a direct transition to the SYN-RCVD state from the LISTEN state. In other words, it shows a number of connections to the local computer, which are initiated by remote computers. The value is a cumulative total.
tcpv4ConnectionsReset	Connections Reset is the number of times TCP connections have made a direct transition to the CLOSED state from either the ESTABLISHED state or the CLOSE-WAIT state.
tcpv4InErrors	The number of errors received.
tcpv4OutResets	The number of segments transmitted with the reset flag set.
tcpv4SegmentsReceived	Segments Received is the number of segments received, including those received in error. This count includes segments received on currently established connections.
tcpv4SegmentsRetransmitted	Segments Retransmitted is the number of segments retransmitted, that is, segments transmitted containing one or more previously transmitted bytes.
tcpv4SegmentsSent	Segments Sent is the number of segments sent, including those on current connections, but excluding those containing only retransmitted bytes.
tcpv6ConnectionFailures	Connections Established is the number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT.
tcpv6ConnectionsActive	Connections Active is the number of times TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state. In other words, it shows a number of connections which are initiated by the local computer. The value is a cumulative total.
tcpv6ConnectionsEstablished	Connection Failures is the number of times TCP connections have made a direct transition to the CLOSED state from the SYN-SENT state or the SYN-RCVD state, plus the number of times TCP connections have made a direct transition to the LISTEN state from the SYN-RCVD state.
tcpv6ConnectionsPassive	Connections Passive is the number of times TCP connections have made a direct transition to the SYN-RCVD state from the LISTEN state. In other words, it shows a number of connections to the local computer, which are initiated by remote computers. The value is a cumulative total.
tcpv6ConnectionsReset	Connections Reset is the number of times TCP connections have made a direct transition to the CLOSED state from either the ESTABLISHED state or the CLOSE-WAIT state.
tcpv6InErrors	The number of errors received.
tcpv6OutResets	The number of segments transmitted with the reset flag set.
tcpv6SegmentsReceived	Segments Received is the number of segments received, including those received in error. This count includes segments received on currently established connections.
tcpv6SegmentsRetransmitted	Segments Retransmitted is the number of segments retransmitted, that is, segments transmitted containing one or more previously transmitted bytes.
tcpv6SegmentsSent	Segments Sent is the number of segments sent, including those on current connections, but excluding those containing only retransmitted bytes.
threadCount	Get the number of threads currently running

Statistic	Description
virtualInUse	The memory currently committed by the system, in bytes. Also called the Commit Total. This is generally equal to the sum of the bytes used of physical memory and paging/swap file(s).
virtualMax	The maximum memory that can be committed by the system without extending the paging file(s), in bytes. Also called the Commit Limit. If the paging/swap file can be extended, this is a soft limit. This is generally equal to the sum of the sizes of physical memory and paging/swap file(s).
xmitBytes	The total number of network bytes transmitted (excluding loopback)
xmitCollisions	The total number of network transmit collisions
xmitDrops	The total number of network transmits dropped
xmitErrors	The total number of network transmit errors
xmitPackets	The total number of network packets transmitted (excluding loopback)

System Processors (ProcessorStats)

Statistics about individual processors on the member's machine. The statistics are:

Statistic	Description
currentFreq	Maximum frequency (in Hz), of the logical processors on this CPU.
processorCpuLoad	CPU usage
processorCpuLoadTicksIDLE	Time spent in Idle
processorCpuLoadTicksIOWAIT	Time spent in IOWait
processorCpuLoadTicksIRQ	Time spent in IRQ
processorCpuLoadTicksNICE	Time spent in Nice
processorCpuLoadTicksSOFTIRQ	Time spent in SoftIRQ
processorCpuLoadTicksSTEAL	Time spent in Steal
processorCpuLoadTicksSYSTEM	Time spent in System
processorCpuLoadTicksUSER	Time spent in User

Linux Networking Statistics (NetworkInterfaceStats)

Statistics about individual network interfaces on the member's machine. The statistics are:

Statistic	Description
bytesReceived	The bytes received.
bytesSent	The bytes sent.
collisions	Packet collisions.
inDrops	Incoming/Received dropped packets.
inErrors	Input errors
mtu	The interface Maximum Transmission Unit (MTU).
outErrors	Output errors

Statistic	Description
packetsReceived	The packets received
packetsSent	The packets sent.
speed	The speed of the network interface in bits per second.

Disk / Filesystem Statistics (DiskStatistics)

Statistics about the disk devices that host filesystems on the member's machine. This does not include any loopback devices or read-only filesystems. The statistics are:

Statistic	Description
flushes	The total number of flush requests completed successfully.
flushesMs	The total number of milliseconds spent by all flush requests.
ioMs	The total number of milliseconds spent doing I/Os.
ioQueueLength	Incremented as requests are given to appropriate struct request_queue and decremented as they finish.
reads	The total number of reads completed successfully.
readsMs	The total number of milliseconds spent by all reads.
writes	The total number of writes completed successfully.
writesMs	The total number of milliseconds spent by all writes.

Partitioned Regions (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 containsKeys completed.
containsKeyOpsRetried	Number of containsKey or containsValueForKey operations which had to be retried due to failures.
containsKeyRetries	Total number of times containsKey or containsValueForKey operations had to be retried.
containsKeyTime	Total time spent performing containsKey operations.
containsValueForKeyCompleted	Number of containsValueForKeys completed.
containsValueForKeyTime	Total time spent performing containsValueForKey operations.
createOpsRetried	Number of create operations which had to be retried due to failures.
createRetries	Total number of times put operations had to be retried.

Statistic	Description
createsCompleted	Number of creates completed.
createTime	Total time spent doing create operations.
destroyOpsRetried	Number of destroy operations which had to be retried due to failures.
destroyRetries	Total number of times destroy operations had to be retried.
destroysCompleted	Number of destroys completed.
destroyTime	Total time spent doing destroys.
getEntryCompleted	Number of getEntry operations completed.
getEntryTime	Total time spent performing getEntry operations.
getOpsRetried	Number of get operations which had to be retried due to failures.
getRetries	Total number of times get operations had to be retried.
getsCompleted	Number of gets completed.
getTime	Total time spent performing get operations.
invalidateOpsRetried	Number of invalidate operations which had to be retried due to failures.
invalidateRetries	Total number of times invalidate operations had to be retried.
invalidatesCompleted	Number of invalidates completed.
invalidateTime	Total time spent doing invalidates.
putOpsRetried	Number of put operations which had to be retried due to failures.
putRetries	Total number of times put operations had to be retried.
putsCompleted	Number of puts completed.
putTime	Total time spent doing puts.

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 PartitionMessages Processed.
partitionMessagesProcessingTime	Total time spent on PartitionMessages processing.
partitionMessagesReceived	Number of PartitionMessages Received.
partitionMessagesSent	Number of PartitionMessages Sent.
prMetaDataSentCount	Total number of times meta data refreshed sent on client's request.

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
actualRedundantCopies	Actual number of redundant copies for this partitioned region.
bucketCount	Total number of buckets in this member.
configuredRedundantCopies	Configured number of redundant copies for this partitioned region.
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.
lowRedundancyBucketCount	Current number of buckets without full redundancy.

Partitioned Region Statistics on Redundancy

These statistics track status on partitioned region data copies. The primary statistics are:

Statistic	Description
actualRedundantCopies	The least current redundant number of copies for any data in this partitioned region (there may be some data that is fully redundant, but some data will have only this number of copies). This value may drop when a 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. Note: If this value remains low, then partitioned region data is at risk and may be lost if another data store is lost. 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, terminating one data store will cause its data to fail over to another data store).
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
lowRedundancyBucketCount	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. If this value remains above zero, then partitioned region data is at risk and may be lost if another data store is lost. This value will be above zero whenever actualRedundantCopies is less than configuredRedundantCopies. A healthy partitioned region will maintain a value of zero. The user should add one or more datstores if this value remains above zero. High-availability may result in a brief fluctuation, but it should return to zero if there are sufficient data stores present (that is, terminating one data store will cause its data to fail over to another data store).

Region Entry Eviction – Count-Based (LRUStatistics)

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.
lruDestroys	Number of entries destroyed in the region through both destroy cache operations and eviction.
lruEvaluations	Number of entries evaluated during LRU operations.
lruEvictions	Number of total entry evictions triggered by LRU.
lruGreedyReturns	Number of non-LRU entries evicted during LRU operations

Region Entry Eviction – Heap-based eviction (HeapLRUStatistics)

The least recently used (LRU) mechanism that keeps the JVM heap size under a given set point records these LRUStatistics. The primary statistics are:

Statistic	Description
entryBytes	The amount of memory currently used by regions configured for eviction.
lruDestroys	Number of entries destroyed in the region through both destroy cache operations and eviction.
lruEvaluations	Number of entries evaluated during LRU operations.
lruEvictions	Number of total entry evictions triggered by LRU.
lruGreedyReturns	Number of non-LRU entries evicted during LRU operations

Region Entry Eviction – Size-based (MemLRUStatistics)

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
byteCount	Number of bytes in region.
bytesAllowed	Number of total bytes allowed in this region.
lruDestroys	Number of entries destroyed in the region through both destroy cache operations and eviction.
lruEvaluations	Number of entries evaluated during LRU operations.
lruEvictions	Number of total entry evictions triggered by LRU.
lruGreedyReturns	Number of non-LRU entries evicted during LRU operations

Server Notifications for All Clients (CacheClientNotifierStatistics)

Statistics regarding cache server operations sent to all clients.

Statistic	Description
clientHealthMonitorRegister	Number of client Register.
clientHealthMonitorUnRegister	Number of client UnRegister.
clientRegistrations	Number of clients that have registered for updates.
clientRegistrationTime	Total time spent doing client registrations.
cqProcessingTime	Total time spent by the cache client notifier processing cqs.
durableReconnectionCount	Number of times the same durable client connects to the server
eventProcessingTime	Total time spent by the cache client notifier processing events.
events	Number of events processed by the cache client notifier.
eventsEnqueuedWhileClientAwayCount	Number of events enqueued in queue for a durable client
queueDroppedCount	Number of times client queue for a particular durable client is dropped

Server Notifications for Single Client (CacheClientProxyStatistics)

Statistics regarding cache server operations and cache server client notifications sent to a single client.

Statistic	Description
cqCount	Number of CQs on the client.
messageProcessingTime	Total time spent sending messages to clients.
messageQueueSize	Size of the message queue.
messagesFailedQueued	Number of client messages attempted but failed to be added to the message queue.
messagesNotQueuedNotInterested	Number of client messages received but not added to the message queue because the client represented by the receiving proxy was not interested in the message's key.
messagesNotQueuedOriginator	Number of client messages received but not added to the message queue because the receiving proxy represents the client originating the message.

Statistic	Description
messagesProcessed	Number of client messages removed from the message queue and sent.
messagesQueued	Number of client messages added to the message queue.
messagesReceived	Number of client messages received.

Server-to-Client Messaging Performance (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
eventsConflated	Number of events conflated for the queue.
eventsExpired	Number of events expired from the queue.
eventsQueued	Number of events added to queue.
eventsRemoved	Number of events removed from the queue.
eventsRemovedByQrm	Number of events removed by QRM message.
eventsTaken	Number of events taken from the queue.
numSequenceViolated	Number of events that has violated sequence.
numVoidRemovals	Number of void removals from the queue.
threadIdentifiers	Number of ThreadIdenfier objects for the queue.

Statistics Collection (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.
delayDuration	Actual duration of sampling delay taken before taking this sample.
jvmPauses	Total number of JVM pauses (which may or may not be full GC pauses) detected by this sampler. A JVM pause is defined as a system event which kept the statistics sampler thread from sampling for 3000 or more milliseconds. This threshold can be customized by setting the system property gemfire.statSamplerDelayThreshold (units are milliseconds).

Transaction Reference Material

This archival topic provides links to reference material for understanding how to work with JTA transaction in VMware Tanzu GemFire.

- JTA Global Transactions with Tanzu GemFire

Use JTA global transactions to coordinate Tanzu GemFire transactions and JDBC transactions.

JTA Global Transactions with Tanzu GemFire

The Java Transaction API, JTA, is a standard Java interface you can use to coordinate VMware Tanzu GemFire transactions and JDBC transactions globally under one umbrella.

Use JTA global transactions to coordinate Tanzu GemFire transactions and JDBC transactions.

JTA provides direct coordination between the Tanzu 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 Tanzu GemFire transaction manager and the JDBC resource manager, responsible for managing operations in the Tanzu GemFire cache and database, respectively

Using JTA, your application controls all transactions in the same standard way, whether the transactions act on the Tanzu GemFire cache, a JDBC resource, or both together. When a JTA global transaction is finished, the Tanzu GemFire transaction and the database transaction are both complete.

When using JTA global transactions with Tanzu GemFire, you have two options:

- Coordinate with an external JTA transaction manager in a container (such as WebLogic or JBoss)
- Set Tanzu GemFire as the “last resource” while using a container (such as WebLogic or JBoss) 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 Tanzu GemFire as usual.

See the Java documentation for more information about topics such as JTA, javax.transaction, committing and rolling back global transactions, and the related exceptions.

- Coordinating with External JTA Transactions Managers

  Tanzu GemFire can work with the JTA transaction managers of several containers like JBoss, WebLogic, GlassFish, and so on.

- Using Tanzu GemFire as the “Last Resource” in a Container-Managed JTA Transaction

  The “last resource” feature in certain third party containers such as WebLogic allow the use one non-XAResource (such as Tanzu GemFire) in a transaction with multiple XAResources while ensuring consistency.

- Behavior of Tanzu GemFire Cache Writers and Loaders Under JTA

  When Tanzu GemFire participates in a global transactions, you can still have Tanzu GemFire cache writers and cache loaders operating in the usual way.

- Turning Off JTA Transactions

  You can configure regions to not participate in any JTA global transaction.

Coordinating with External JTA Transaction Managers

Tanzu GemFire can work with the JTA transaction managers of several containers such as JBoss, WebLogic, GlassFish, and so on.

At startup Tanzu GemFire looks for a TransactionManager (javax.transaction.TransactionManager) that has been bound to its JNDI context. When Tanzu GemFire finds such an external transaction manager, all Tanzu 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 Tanzu 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 Tanzu GemFire transaction is started automatically, and we register a javax.transaction.Synchronization callback with the external JTA transaction manager.
2. At transaction commit, Tanzu GemFire gets a beforeCommit() callback from the external JTA transaction manager. Tanzu GemFire does all locking and conflict detection at this time. If this fails, an exception is thrown back to JTA transaction manager, which then cancels the transaction.
3. After a successful beforeCommit()callback, JTA transaction manager asks other data sources to commit their transaction.
4. Tanzu GemFire then gets a afterCommit() callback in which changes are applied to the cache and distributed to other members.

You can deactivate JTA in any region that should not participate in JTA transactions. See Turning Off JTA Transactions.

How to Run a JTA Transaction Coordinated by an External Transaction Manager

Use the following procedure to run a Tanzu 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 Tanzu GemFire for any necessary transactional behavior in the cache.xml file. For example, enable copy-on-read and specify a transaction listener, as needed. See Copy on Read Behavior.

2. Make sure that JTA transactions are enabled for the regions that will participate in the transaction. See Turning Off JTA Transactions for details. 4.

Start the transaction through the external container.

3. Initialize the Tanzu GemFire cache. Tanzu GemFire will automatically join the transaction. 6.

   Execute operations in the cache and the database as usual.

4. Commit the transaction through the external container.

Using Tanzu GemFire as the “Last Resource” in a Container-Managed JTA Transaction

The “last resource” feature in certain third party containers such as WebLogic allow the use of one non-XAResource (such as Tanzu GemFire) in a transaction with multiple XAResources while ensuring consistency.

In the previous two JTA transaction use cases, if the Tanzu GemFire member fails after the other data sources commit but before Tanzu GemFire receives the afterCommit callback, Tanzu GemFire and the other data sources may become inconsistent. To prevent this from occurring, you can use the container’s “last resource optimization” feature, with Tanzu GemFire set as the “last resource”. Using Tanzu GemFire as the last resource ensures that in the event of failure, Tanzu GemFire remains consistent with the other XAResources involved in the transaction.

To accomplish this, the application server container must use a JCA Resource Adapter to accomodate Tanzu GemFire as the transaction’s last resource. The transaction manager of the container first issues a “prepare” message to the participating XAResources. If the XAResources all accept the transaction, then the manager issues a “commit” instruction to the non-XAResource (in this case, Tanzu GemFire). The non-XAResource (in this case, Tanzu GemFire) participates as a local transaction resource. If the non-XAResource fails, then the transaction manager can rollback the XAResources.

How to Run JTA Transactions with Tanzu GemFire as a “Last Resource”

1. Locate the version-specific gemfire-jca RAR file within the lib directory of your Tanzu GemFire installation.
2. Add your container-specific XML file to the gemfire-jca RAR file.

1. Create a container-specific resource adapter XML file named <container>-ra.xml. For example, an XML file for a WebLogic resource adapter XML file might look something like this: ```pre GFE JCA gfe/jca ```
2. Create a folder named `META-INF`, and place the container-specific XML file inside the directory. For example, the folder structure would look like this: ```pre META-INF/weblogic-ra.xml ```
3. Navigate to the directory above the `META-INF` folder and execute the following command, with appropriate substitutions for path and file names: ```pre $ jar -uf /path/to/lib/gemfire-jca-X-X-X.rar META-INF/weblogic-ra.xml ```

1. Make sure that the gemfire-dependencies.jar is accessible in the class path of the JTA transaction coordinator container.
2. Deploy the version-specific gemfire-jca RAR file on the JTA transaction coordinator container. When deploying the file, you specify the JNDI name and so on.
3. Configure Tanzu GemFire for any necessary transactional behavior. Enable copy-on-read and specify a transaction listener, if you need one. See Copy on Read Behavior.
4. Get an initial context through org.apache.geode.cache.GemFireCache.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.
5. Start and commit the global transaction using the UserTransaction object rather than with Tanzu GemFire's CacheTransactionManager.

   ```
   UserTransaction txManager = (UserTransaction)ctx.lookup("java:/UserTransaction");
   ```
6. Obtain a Tanzu GemFire connection.

   ```
   GFConnectionFactory cf = (GFConnectionFactory) ctx.lookup("gfe/jca");

   //This step of obtaining connection is what begins the
   //LocalTransaction.
   //If this is absent, GFE operations will not be part of any
   //transaction
   GFConnection gemfireConn = (GFConnection)cf.getConnection();
   ```

Behavior of GemFire Cache Writers and Loaders Under JTA

This topic explains the behavior of VMware Tanzu GemFire cache writers and loaders Under JTA.

When Tanzu GemFire participates in a global transactions, you can still have Tanzu 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. For more on transactional data sources, see the discussion of XAPooledDataSource and ManagedDataSource in Configuring Database Connections Using JNDI.

If you use a Tanzu 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

This topic explains how to 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.

Experimental Features

This topic provides links to documents about “experimental” features in VMware Tanzu GemFire. Tanzu GemFire includes these experimental features to allow interested developers to experiment with their use in their own development systems.

Feedback will help drive development of these features, and they may be supported in future versions of the product.

These features may change in or be removed from future versions of the product.

- Cluster Management Service

  The cluster management service enables you to change the configuration of the cluster (for example, create/destroy regions, indexes, or gateway receivers/senders), and have these changes replicated on all the applicable servers and persisted in the cluster configuration service.

- Automated Rebalancing of Partitioned Region Data

  The automated rebalance feature triggers a rebalance operation based on a time schedule.

Cluster Management Service

This topic explains the VMware Tanzu GemFire cluster management service.

> This feature is experimental and is subject to change in future releases of Tanzu GemFire.

Use the cluster management service to:

- Change the configuration of a cluster using a REST interface.
- Replicate the changes to all the applicable servers.
- Persist the changes in the cluster configuration service.

This REST interface differs from gfsh commands in the following ways:

- The REST interface allows for a more programmable access.
- Changes to all members of the cluster are applied consistently.
- REST interface changes may be invoked from a server via a function or from any client application.
- You can configure the cluster without having any servers running. Having a locator running is sufficient.
- Not all gfsh commands and options are available with this REST interface.

Currently, Tanzu GemFire provides a Java client to interact with the REST interface. There is also a Golang client that interacts with the REST interface.

The cluster management service REST API adheres to standard REST semantics, so users can use POST, PATCH, DELETE, and GET to create, update, delete, and read, respectively.

This version of Tanzu GemFire uses the 1.15.0 Management REST API.

Cluster Management Configuration

Locators started by gfsh support the cluster management service REST API.

Three property settings must be in effect to enable the cluster management REST API. These are in effect, by default, for gfsh-initiated locators.

- http-service-port must be set to a non-zero value. If set to zero, no HTTP service is available for the locator. The default HTTP service port is 7070.
- enable-cluster-configuration must be set to true. When set to false, all of the cluster management service REST API calls return errors.
- enable-management-rest-service must be set to true. When set to false, the cluster management REST service does not start.

By default, cluster management REST service logging occurs at the debug level. To increase logging to the info level, set the system property -Dgeode.management.request.logging=true when you start up locators.

Authentication and Authorization

Secure your cluster management service REST API by implementing the SecurityManager interface. When the interface is implemented, API requests require credentials.

Credentials can be a username/password pair or a valid bearer token. By default, the cluster management service expects credentials in the form of a username and password, for example:

```
curl --user username:password http://example.com/management/regions
```

When the security-auth-token-enabled-components property is set to all or management, the cluster management service expects credentials in the form of a valid bearer token. For example:

```
curl -H "Authorization: Bearer Y6bXlwYXNzdzByZAo" http://example.com/management/regions
```

Java API

The Java client allows you to use the REST interface API in your Java application. To use the API:

- Include gemfire-management.jar in your application's classpath.
- Use the ClusterManagementServiceBuilder class to build a ClusterManagementService object, using the various methods available on the builder to set the connection configuration.

Example:

```
ClusterManagementService cmsClient = new ClusterManagementServiceBuilder()
    .setHost("localhost")
    .setPort("7070")
    .setTransport(new MyCustomTransport())
    .build();
Region regionConfig = new Region();
ClusterManagementListResult<Region, RuntimeRegionInfo> list = cmsClient.list(region);
```

> In Tanzu GemFire 10.2 and later, an implementation of ClusterManagementServiceTransport is required. It must be provided via the setTransport method or be available on the

classpath for service loading.

Golang Command Line Interface

A Golang client that interacts with this service is available at https://github.com/gemfire/tanzu-gemfire-management-cf-plugin.

After downloading, you can use it to interact with any running cluster. For example:

```
$ ./gemfire http://localhost:7070 list regions
```

Automated Rebalance

This topic explains automated rebalancing of partitioned region data in VMware Tanzu GemFire.

> This feature is experimental and is subject to change in future releases of Tanzu GemFire.

Automated rebalance triggers a rebalance (see Rebalancing Partitioned Region Data) operation based on a time schedule. At the scheduled intervals, the balance of the partitioned regions is evaluated based on configured criteria. One criterion is a minimum threshold for number of bytes that would be transferred if the rebalance takes place. The other criterion uses the ratio of the number of bytes that would be transferred to the total number of bytes in the regions. If the evaluation indicates the system is out of balance, the rebalance transfer is initiated.

To enable automated rebalance, register the rebalance manager as an initializer either with the `<initializer>` attribute in the `<cache>` configuration of the `cache.xml` file:

```
<class-name> org.apache.geode.cache.util.AutoBalancer </class-name>
```

Or the `create initializer` gfsh command:

```
create initializer --initializer=org.apache.geode.cache.util.AutoBalancer
```

The time schedule that triggers an evaluation and possible rebalance uses a cron-based specification in the `<initializer>` attribute within the `<cache>` configuration of the `cache.xml` file. If using gfsh to configure the initializer, you can provide parameters in JSON format after your class name via the `--initializer` option. This scheduling specification is required. Specify the cron expression in the Spring format. This example specification triggers each Saturday at 3am:

```
<parameter name="schedule"> 0 0 3 ? * SAT </parameter>

create initializer --initializer='org.apache.geode.cache.util.AutoBalancer{"schedule":"0 0 3 ? * SAT"}'
```

> The single quotes around the value of `--initializer` are required because the cron expressions contain spaces.

This example specification triggers once each day at 4am:

```
<parameter name="schedule"> 0 0 4 * * ?</parameter>

create initializer --initializer='org.apache.geode.cache.util.AutoBalancer{"schedule":"0 0 4 * * ?"}'
```

The automated rebalance specifications that specify criteria for triggering the rebalance are optional and have reasonable default values.

One criterion is a minimum number of bytes that would be transferred if the rebalance were to take place. The specification is in units of bytes. This example shows the specification for the default value of 100 MB:

```
<parameter name="minimum-size"> 104857600 </parameter>

create initializer --initializer='org.apache.geode.cache.util.AutoBalancer{...,"minimum-size":"104857600"}'
```

Another criterion represents the ratio of bytes that would be transferred to the total number of bytes in the partitioned regions, represented as an integer percentage. The default is 10 percent. This example specifies 15 percent:

```
<parameter name="size-threshold-percent"> 15 </parameter>

create initializer --initializer='org.apache.geode.cache.util.AutoBalancer{...,"size-threshold-percent":"15"}'
```

Glossary

This glossary defines terms used in VMware Tanzu GemFire documentation.

ACK wait threshold

A time-to-wait for message acknowledgment between system members.

administrative event

See event.

API

Application Programming Interface. Tanzu GemFire provides APIs to cached data for Java applications.

application program

A program designed to perform a specific function directly for the user or, in some cases, for another application program. Tanzu GemFire applications use the Tanzu GemFire application programming interfaces (APIs) to modify cached data.

attribute

Querying: A named member of a data object. The public fields and methods of an object may be accessed as attributes in the context of a query.

Region: See region attributes.

attribute path

A sequence of attributes separated by a dot (.), applied to objects where the value of each attribute is used to apply the next attribute.

blocking

A behavior associated with synchronization functions. Blocking behavior is exhibited as waiting for a signal to proceed, regardless of how long it takes. See also timeout.

cache

In-memory Tanzu 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 cluster. See also local and remote.

cache-local

Residing or occurring in the local cache.

cache.xml

Common name for the XML file that declares the initial configuration of a cache. This file is used to customize the behavior of the Tanzu GemFire cache server process and can be used by any Java application. Applications can also configure the cache through the Tanzu GemFire Java APIs. You can give this file any name.

cache event

See event.

cache listener

User-implemented plug-in for receiving and handling region entry events. A region's cache listener is called after an entry in the local cache is modified. See also cache writer.

cache loader

User-implemented plug-in for loading data into a region. A region's cache loader is used to load data that is requested of the region but is not available in the cluster. For a distributed region, the loader that is used can be in a different cache from the one where the data-request operation originated. See also netSearch and netLoad.cache misses, where a requested key is not present or has a null value in the local cache.

cache miss

The situation where a key's value is requested from a cache and the requested key is not present or has a null value. Tanzu 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 Tanzu GemFire cluster member process that can service client connections.

cache transaction

A native Tanzu GemFire transaction, managed by Tanzu GemFire and not by JTA. This type of transaction operates only on data available from the Tanzu GemFire cache in the local member. See also JTA and global transaction.

cache writer

User-implemented plug-in intended for synchronizing the cache with an outside data source. A region's cache writer is a synchronous listener to cache data events. The cache writer has the ability to cancel a data modification. See also cache listener and netWrite.

client

A Tanzu GemFire application that is configured as a standalone cluster member, with regions configured as client regions. Client configuration uses the <client-cache> cache.xml element and the ClientCache API.

client region

A Tanzu GemFire cache region that is configured to go to one or more Tanzu GemFire servers, in a separate Tanzu GemFire cluster, 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.

cluster configuration service

The cluster configuration service saves cluster configurations created by gfsh commands to the locators in a cluster and distributes the configurations to members of the cluster.

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.

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 Tanzu GemFire, region entry update events, server events going to clients, and gateway sender events going to remote clusters can all be conflated.

connection

The connection used by an application to access a Tanzu GemFire system. A Java application connects to its Tanzu GemFire cluster when it creates its cache. The application must connect to a cluster to gain access to the Tanzu GemFire functionalities. A client connects to a running Tanzu 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

Tanzu 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 cluster arrive into the local cache. Client consumers can register subscriptions with their servers so that updates are automatically forwarded from the server tier. See producer.

coordinator

The member of the cluster that sends out membership views. This is typically the locator in Tanzu 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.

data entry

See entry.

data fabric

Also referred to as an in-memory data grid or an enterprise data fabric. Tanzu 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-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 cluster among the data stores. See also data accessor.

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 Tanzu GemFire system members or clusters 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 (Deprecated - no replacement)

Data distribution setting that causes asynchronous distribution operations, which return without waiting for a response from other caches. This scope produces the best performance, but is prone to race conditions.

entry

A data object in a region consisting of a key and a value. The value is either null (invalid) or a Java object. A region entry knows what region it is in. An entry can have an application-defined user attribute. See also region data, entry key, and entry value.

entry key

The unique identifier for an entry in a region.

entry value

The data contained in an entry.

event

An action recognized by the Tanzu GemFire system members, which can respond by executing callback methods. The Tanzu 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 cluster 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 cluster. 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 Tanzu 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 cluster site and an associated queue for distributing region events in a multi-site deployment.

gemfire.properties

Common name for the file used for cluster configuration, including system member connection and communication behavior, logging and statistics files and settings, and security settings. Applications can also configure the cluster through the Tanzu GemFire Java APIs. You can give this file any name.

global scope

Data distribution setting that provides locking across the cluster 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 cluster.

global transaction

A JTA-controlled transaction in which multiple resources, such as the Tanzu GemFire cache and a JDBC database connection, participate. JTA coordinates the completion of the transaction with each of the transaction's resources. See also JTA and cache transaction.

HTTP

World Wide Web's 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 and expiration action.

initial capacity

Region attribute. The initial capacity of the map used for storing region entries.

invalid

The state of an object when the cache holding it does not have the current value of the object.

invalidate

Distributed: To mark an object as being invalid across the distributed cache.

Local: To mark an object as being invalid in the local cache only.

JDBC

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. Tanzu 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. Tanzu GemFire can be a member of a JTA global transaction. See also global transaction.

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 cluster or in another cluster. See also remote.

Region with local scope: A region whose scope is set to local. This type of region does not distribute anything with other members in the cluster.

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

Tanzu 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. .

LRU

Least recently used. Used to refer to region entry or entries most eligible for eviction due to lack of interest by client applications. Tanzu 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.

machine

Any Tanzu GemFire-supported physical machine or Virtual Machine.

member

A process that has defined a connection to a Tanzu GemFire cluster and created a Tanzu GemFire cache. This can be a Java or Native Client application. This can also be a Tanzu GemFire process such as a locator or cacheserver. The minimal Tanzu GemFire process configuration is a single member that is connected to a cluster.

message queue

A first-in, first-out data structure in a Tanzu 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 cluster, a client and server, or two members in different clusters. See also conflation.

mirroring

See replicate.

named region attributes

Region attributes that are stored in the member memory and can be retrieved through their region attributes refid setting. Tanzu 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 Tanzu 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 Tanzu 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 Tanzu 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. Tanzu GemFire supports a subset of OQL.

off-heap memory

Memory that is not on the standard Java heap and that is not managed by the JVM and its garbage collector.

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 Tanzu 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 clusters.

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 Tanzu GemFire member process that puts data into the cache for consumption by other members. Producers may be configured with empty regions, where the data they put into the cache is not stored locally, but causes cache update events to be sent to other members. This is a common configuration in peer members and for client processes. See consumer.

pull model

Data distribution model where each process receives updates only for the data in which the process has explicitly expressed interest. In a Tanzu 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 Tanzu GemFire client, you get pull behavior by specifically subscribing to the entries of interest. See push model.

push model

Data distribution model where each process receives updates for everything in the data set. In a Tanzu 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 Tanzu GemFire client, you get push behavior by registering interest in all keys in the region. See pull model.

query string

A fully-formed SQL statement that can be passed to a query engine and executed against a data set. A query string may or may not contain a SELECT statement.

race condition

Anomalous behavior caused by the unexpected dependence on the relative timing of events. Race conditions often result from incorrect assumptions about possible ordering of events.

range-index

An XPath index optimized for range-queries with the added index maintenance expense of sorting the set of values. A range index allows faster retrieval of the set of nodes with values in a certain range. See also structure-index and value-index.

region

A logical grouping of data within a cache. Regions usually contain data entries (see entry). Each region has a set of region attributes governing activities such as expiration, distribution, data loading, events, and capacity control. In addition, a region can have an application-defined user attribute.

region attributes

The class of attributes governing the creation, distribution, and management of a region and its entries.

region data

All of the entries directly contained in the region.

region entry

See entry.

region shortcut

Enums RegionShortcut and ClientRegionShortcut defining the main region types in Tanzu 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 Tanzu 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 cluster. 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 cluster. In multi-site installations, a gateway sends events from the local cache to remote caches in other clusters. See also local.

replicated region

A region with data-policy set to replicate or persistent-replicate.

replicate

Region data-policy specification indicating to copy all distributed region data into the local cache at region creation time and to keep the local cache consistent with the distributed region data.

resource manager

Tanzu 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 and eviction-attributes.

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.

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 Tanzu GemFire member application that is configured as a peer in its own system and as a server to connecting Tanzu GemFire client applications.

server group

An optional logical grouping of servers in a server clusters. There is always the default server group made up of all available server in the server clusters. 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. 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 cluster configured for no communication with peers. Client applications are generally defined with standalone clusters, 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 and value-index.

system member

See member.

TCP

The Transmission Control Protocol is a part of the internet protocol (IP) suite that provides reliable connection-oriented communication. 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.

timeout

A behavior associated with synchronization functions. Timeout behavior is exhibited as refusal to wait longer than a specified time for a signal to proceed. See also blocking.

time-to-live

The amount of time a region or region entry may remain in the cache without being modified before being expired. Entry modification includes creation, update, and removal. Region modification includes creation, update, or removal of the region or of any of its entries.

Time-to-live attributes are set at the region level for the region and at the entry level for entries. See also idle timeout and expiration action.

transaction

See cache transaction and global transaction.

transaction listener

User-implemented plug-in for receiving and handling transaction events. A transaction listener is called after a transaction commits. See also transaction writer.

transaction writer

User-implemented plug-in intended for synchronizing the cache with an outside data source. A transaction writer is a synchronous listener to cache transactions. The transaction writer has the ability to veto a transaction. See also transaction listener.

transactional view

The result of a history of transactional operations for a given open transaction.

transport layer

The network used to connect the Tanzu GemFire system members in a Tanzu GemFire system.

TTL

See time-to-live.

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. Tanzu GemFire does not use these attributes. Compare to region attributes, which are used by Tanzu GemFire.

value constraint

Enforcing a specific entry value type. The value-constraint region attribute, when set, constrains the entries in the region to values of the specified object type. Value constraints can be used to provide object typing for region querying and indexing. The value-constraint is only checked in the cache that does the

entry put or create operation. When the entry is distributed to other caches, the value constraint is not checked.

value-index

An XPath index that operates much as a structure-index does, but that separates the nodes that match the XPath expression into sets mapped by each node's value. This allows further filtering of the nodes to be evaluated in a query by going directly to those with a specific value. See also structure-index and range-index.

view

A collection of member identifiers that defines the membership group.

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 it own virtual (ie, software-based) CPU, RAM hard disk and network interface card (NIC). An operating system cannot 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 XSD (XML schema definition.)

XML schema definition

The definition of the structure, content, and semantics used in an XML document. The definition can be used to verify that each item of content in a document adheres to the specification of the element in which the content is placed. 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.

XSD

See XML schema definition.

Documentation Legal Notice

Information about the documentation legal notice.

This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by Broadcom at any time. This Documentation is proprietary information of Broadcom and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of Broadcom.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all Broadcom copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to Broadcom that all copies and partial copies of the Documentation have been returned to Broadcom or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, BROADCOM PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL BROADCOM BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF BROADCOM IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice

The manufacturer of this Documentation is Broadcom Inc.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2005–2025 Broadcom. All Rights Reserved. The term “Broadcom” refers to Broadcom Inc. and/or its subsidiaries. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.