
GemStone®

System Administration Guide for GemStone/S 64 Bit™

Version 3.4

October 2016



INTELLECTUAL PROPERTY OWNERSHIP

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

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

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

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

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

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

COPYRIGHTS

This software product, its documentation, and its user interface © 1986-2016 GemTalk Systems LLC. All rights reserved by GemTalk Systems.

PATENTS

GemStone software is covered by U.S. Patent Number 6,256,637 “Transactional virtual machine architecture”, Patent Number 6,360,219 “Object queues with concurrent updating”, Patent Number 6,567,905 “Generational garbage collector with persistent object cache”, and Patent Number 6,681,226 “Selective pessimistic locking for a concurrently updateable database”. GemStone software may also be covered by one or more pending United States patent applications.

TRADEMARKS

GemTalk, **GemStone**, **GemBuilder**, **GemConnect**, and the GemTalk logo are trademarks of GemTalk Systems LLC, or of VMware, Inc., previously of GemStone Systems, Inc., in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Solaris, **Java**, and **Oracle** are trademarks or registered trademarks of Oracle and/or its affiliates. **SPARC** is a registered trademark of SPARC International, Inc.

Intel and **Pentium** are registered trademarks of Intel Corporation in the United States and other countries.

Microsoft, **Windows**, and **Windows Server** are registered trademarks of Microsoft Corporation in the United States and other countries.

Linux is a registered trademark of Linus Torvalds and others.

Red Hat and all Red Hat-based trademarks and logos are trademarks or registered trademarks of Red Hat, Inc. in the United States and other countries.

Ubuntu is a registered trademark of Canonical Ltd., Inc., in the U.S. and other countries.

SUSE is a registered trademark of Novell, Inc. in the United States and other countries.

AIX, **POWER6**, **POWER7**, and **POWER8** and **VisualAge** are trademarks or registered trademarks of International Business Machines Corporation.

Apple, **Mac**, **MacOS**, and **Macintosh** are trademarks of Apple Inc., in the United States and other countries.

CINCOM, **Cincom Smalltalk**, and **VisualWorks** are trademarks or registered trademarks of Cincom Systems, Inc.

Other company or product names mentioned herein may be trademarks or registered trademarks of their respective owners. Trademark specifications are subject to change without notice. GemTalk Systems cannot attest to the accuracy of all trademark information. Use of a term in this documentation should not be regarded as affecting the validity of any trademark or service mark.

GemTalk Systems LLC
15220 NW Greenbrier Parkway
Suite 240
Beaverton, OR 97006



Preface

About This Manual

This manual describes the GemStone/S 64 Bit™ repository, and how to setup, tune, monitor, and maintain your GemStone server.

This manual is intended for users that are at least somewhat familiar with using Smalltalk and the Topaz programming environment to execute GemStone Smalltalk code. It also assumes some familiarity with UNIX.

You should have the GemStone system installed on your host computer, as described in the *GemStone/S 64 Bit Installation Guide* for your platform.

For questions or to submit feedback on this manual, join the documentation mailing list: <http://lists.gemtalksystems.com/mailman/listinfo/documentation>.

Terminology Conventions

The term “GemStone” is used to refer to the server products GemStone/S 64 Bit and GemStone/S, and the GemStone family of products; the GemStone Smalltalk programming language; and may also be used to refer to the company, now GemTalk Systems, previously GemStone Systems, Inc. and a division of VMware, Inc.

Typographical Conventions

This document uses the following typographical conventions:

- ▶ Smalltalk methods, GemStone environment variables, operating system file names and paths, listings, and prompts are shown in `monospace` typeface.
- ▶ Responses from GemStone commands are shown in an underlined typeface.
- ▶ Place holders that are meant to be replaced with real values are shown in *italic* typeface.

- ▶ Optional arguments and terms are enclosed in [square brackets].
- ▶ Alternative arguments and terms are separated by a vertical bar (|).

Executing the Examples

The GemStone server is not accessed directly; you must login using Topaz, GemBuilder, or another interface in order to execute code. Topaz is GemStone's command-line interface, and the examples in this manual are primarily presented as executable Smalltalk code in Topaz.

Most of these examples may also be executed in GemBuilder or another interface to GemStone; some exceptions are noted in the text. Some details of the display of returned values may vary depending on the interface used.

Refer to the *Topaz Programming Environment* for more information on Topaz, including establishing a Topaz login and entering and executing commands.

Technical Support

Support Website

gemtalksystems.com

GemTalk's website provides a variety of resources to help you use GemTalk products:

- ▶ **Documentation** for the current and for previous released versions of all GemTalk products, in PDF form.
- ▶ **Product download** for the current and selected recent versions of GemTalk software.
- ▶ **Bugnotes**, identifying performance issues or error conditions that you may encounter when using a GemTalk product.
- ▶ **TechTips**, providing information and instructions that are not in the documentation.
- ▶ **Compatibility matrices**, listing supported platforms for GemTalk product versions.

We recommend checking this site on a regular basis for the latest updates.

Help Requests

GemTalk Technical Support is limited to customers with current support contracts. Requests for technical assistance may be submitted online (including by email), or by telephone. We recommend you use telephone contact only for urgent requests that require immediate evaluation, such as a production system down. The support website is the preferred way to contact Technical Support.

Website: **techsupport.gemtalksystems.com**

Email: **techsupport@gemtalksystems.com**

Telephone: **(800) 243-4772 or (503) 766-4702**

Please include the following, in addition to a description of the issue:

- ▶ The versions of GemStone/S 64 Bit and of all related GemTalk products, and of any other related products, such as client Smalltalk products, and the operating system and version you are using.
- ▶ Exact error message received, if any, including log files and statmonitor data if appropriate.

Technical Support is available from 8am to 5pm Pacific Time, Monday through Friday, excluding GemTalk holidays.

24x7 Emergency Technical Support

GemTalk offers, at an additional charge, 24x7 emergency technical support. This support entitles customers to contact us 24 hours a day, 7 days a week, 365 days a year, for issues impacting a production system. For more details, contact GemTalk Support Renewals.

Training and Consulting

GemTalk Professional Services provide consulting to help you succeed with GemStone products. Training for GemStone/S is available at your location, and training courses are offered periodically at our offices in Beaverton, Oregon. Contact GemTalk Professional Services for more details or to obtain consulting services.



Table of Contents

<i>Chapter 1. Administration of the GemStone/S Environment</i>	23
1.1 Basic GemStone/S 64 Bit Architecture	24
1.2 Starting GemStone and Logging In	25
Stone	25
Shared Page Cache	26
Shared Cache Size.	26
OS memory and Swapping	27
NetLDI.	27
Logging in Gem Sessions	27
Linked vs. RPC	28
GemServers for Large Configurations	28
NRS (Network Resource String)	28
1.3 Authentication and Authorization	29
GemStone UserIds and login	29
Authorization to Access Data	29
File access and authorization	29
1.4 Transactions and commit records	30
Object views.	30
Session transactional state	30
Commit records.	30
1.5 Files and Directories	31
GemStone Installation	31
GemStone shared libraries	31
Lock file directory	31
Host Identifier.	32
Extents, Tranlogs, and disk space.	32
Disk Usage for Extents and Transaction Logs.	32
Disk configuration	32
Raw Partitions.	33

Process Log Files	33
System Clock and GemStone times	33
TimeZones	33
1.6 Options for Configuring	33
Configuration files.	34
Environment variables	34
Utility command arguments	35
Run-time configurations	35
Example Configurations	35
1.7 Avoiding risk of Data loss	36
Recovery vs. Restore	37
Developing a Failover Strategy	37
Verifying strategy	37
1.8 Running a Second Repository	38

Chapter 2. Configuring the GemStone Server **39**

2.1 Server Components	39
The Server Configuration File	40
Before you begin.	40
Number of User Sessions.	41
Shared Page Cache	41
Estimating the Size of the Shared Page Cache	41
Verifying OS support for sufficient Shared Memory	42
2.2 Configuring Extents and Transaction Logs	43
Recommendations About Disk Usage	43
Raw Partitions	43
NFS	44
Configuring the Repository Extents.	44
Estimating Extent Size.	44
Choosing the Extent Location	45
Extent disk configuration.	45
Setting a Maximum Size for an Extent	45
Pregrowing Extents to a Fixed Size	45
Pregrowing Extents to the Maximum Size	46
Allocating Data to Multiple Extents	47
Configuring the Transaction Logs.	49
Logging Mode	49
Estimating Disk Space for Transaction Logs	50
Choosing the Log Location and Size Limit	51
How To Set Up a Raw Partition	52
Sample Setup for Extent on Raw Partition	53
Changing Between Files and Raw Partitions.	53
Moving an Extent to a Raw Partition	53
Moving an Extent to the File System	54
Moving Transaction Logging to a Raw Partition.	54

Moving Transaction Logging to the File System	54
Server Response to Gem Fatal Errors.	55
2.3 How To Access the Server Configuration at Run Time	55
To Access Current Settings at Run Time	55
To Change Settings at Run Time	56
2.4 Tuning Server Performance.	58
Tuning the Shared Page Cache	58
Adjusting the Cache Size.	58
Matching Spin Lock Limit to Number of Processors	58
Clustering Objects That Are Accessed Together.	58
Controlling Checkpoint Frequency.	58
Tuning Page Server Behavior	59
To Add AIO Page Server threads	59
Free Frame Page Server	60
Process Free Frame Caches	60
Running Cache Warming	61
Configure what is loaded into the cache	61
Configure when warming is run	62

Chapter 3. Configuring Gem Session Processes 63

3.1 Overview	63
Linked and RPC Applications.	64
Note on terminology	65
The Session Configuration File	65
3.2 Configuring Gem Session Processes	65
Local vs. Remote	65
Local	65
Remote	66
Gem Memory Requirements	66
Additional Configuration for Remote Gems	66
3.3 Set the Gem Configuration Options	67
Configure Temporary Object Space	67
Configure SSL for remote Gem sessions	67
Private Page Cache	67
Native Code	68
3.4 How To Access the Configuration at Run Time	68
To Access Current Settings at Run Time.	68
To Change Settings at Run Time	68

Chapter 4. NetLDI and Interprocess Access 71

4.1 Overview	71
Administrative user account	72
Login Parameters and Gem process	72

4.2 The NetLDI	73
NetLDI Ports and Names.	73
4.3 NetLDI configuration.	74
Configuration Decisions	75
NetLDI behavior: who will own spawned processes?.	75
Authentication: which requests need to be authenticated?	76
Setting up the NetLDI Configuration	78
1. Setting up NetLDI as root with Authentication	78
Starting the NetLDI	78
Login parameters	78
Using Kerberos authentication.	79
2. Setting up NetLDI in Guest Mode with Captive Account.	79
Starting the netldi	79
Login	79
4.4 File Permissions	80
Using the Setuid Bit	80
Alternative: Use Group Write Permission	81
Shared Page Cache	81
File Permissions for Other Files and Directories.	81
4.5 Linked Gem Sessions	82

Chapter 5. Connecting Distributed Systems **83**

5.1 Overview	83
Distributed configuration with Local Gem	84
Distributed configuration with Remote Gem.	84
Network	85
GemStone NetLDIs	85
NRS Syntax.	85
Stone	86
Connecting to the RPC Gem	86
Shared Page Cache	86
GemStone Page Servers.	87
Port use in GemStone.	87
Disrupted Communications	88
5.2 Configuring GemStone on Remote Nodes	88
Local Gems only.	88
Remote Gems	88
Configuration Examples	89
RPC Application on a Remote Node with Local Gem	90
Linked Application on a Remote Node	91
RPC Application on a Remote Node with Remote Gem	92
RPC Application, Gem, and Stone on Three Nodes	93
Distributed System with a Mid-Level Cache.	94
5.3 Troubleshooting Remote Logins	96
How the Login Process starts Session Processes.	97

RPC Application with Gem on Server Node.	97
Linked Application on Remote Node.	98
If You Still Have Trouble	99
Check NetLDI Log Files	100
5.4 Tuning for Distributed Configurations	101

Chapter 6. Running GemStone **103**

6.1 Starting the GemStone Server.	103
To Start GemStone	104
To Troubleshoot Stone Startup Failures	104
Missing or Invalid Key File	105
Shared Page Cache Cannot Be Attached	105
Extent Missing or Access Denied	106
Extent Open by Another Process	106
Extent Already Exists.	106
Other Extent Failures.	107
Transaction Log Missing.	107
Other Startup Failures	108
Listing Running Servers	108
6.2 Starting a NetLDI	109
To Troubleshoot NetLDI Startup Failures	109
6.3 Starting a GemStone Session	110
To Define a GemStone Session Environment	110
To Start a Linked Session	111
To Start an RPC Session	112
To Troubleshoot Session Login Failures	113
Identifying and Stopping Logged-in Sessions.	114
6.4 Shutting Down Sessions, the Object Server, and NetLDI	116
Stopping Logged-in Sessions	116
Stopping the Stone	116
Stopping the NetLDI	117
Using OS kill	117
6.5 Recovering from an Unexpected Shutdown	117
Clean Shutdown Message	118
Disk Failure or File System Corruption	118
Shared Page Cache Error	119
Fatal Error Detected by a Gem	119
Some Other Shutdown Message	119
No Shutdown Message	120

Chapter 7. Monitoring GemStone **121**

7.1 GemStone Process Logs	122
Finding log files.	122

Stone Log	122
Shared Page Cache Monitor Log	123
Admin Gem Log	124
Reclaim Gem Log	124
Free Frame Page Server Log	125
AIO Page Server Log	125
Page Manager Log	126
Symbol Gem Log	126
NetLDI Log	126
Gem Logs and logs related to Gem Sessions	127
Linked Gems	127
RPC Gems on Stone's host	127
Remote RPC or Linked Gems	127
Further control over log file location and name	128
Logsender and logreceiver Logs	128
Other Log Files	129
Summary of GemStone Process Log Behaviors	129
Managing log files.	131
Retaining or deleting all log files.	131
Customizing individual process deletion behavior	131
Localizing timestamps in log files	131
Programmatically adding messages to logs	132
7.2 Repository Page and Object Audit	132
Page Audit	132
Object Audit and Repair	134
Performing the Object Audit	134
Error Recovery	135
7.3 Profiling Repository Contents	136
7.4 Monitoring Performance	138
Statmonitor and VSD	138
Programmatic Access to Cache Statistics	138
Host Statistics	143
Host Statistics for processes	143
Host Statistics for OS	144

Chapter 8. User Accounts and Security

145

8.1 GemStone Users	145
UserProfiles	145
AllUsers	146
Special System Users	146
UserProfile Data	147
User ID	148
Authentication Scheme	148
Password	148
Default Object Security Policy	149

Privileges	149
Symbol Lists	151
DeletedUserProfile and AllDeletedUsers	152
8.2 UserProfileGroups	153
AllGroups	153
Groups for object authorization.	154
Create a group	154
Delete a group	154
8.3 Creating and Removing Users	154
Creating Users	155
Removing Users	155
Users and Group membership	156
Adding a user to a group	156
Removing a user from a group	157
Querying for Group members.	157
Querying for a User's Groups	157
8.4 Administering Users.	158
List Existing Users	158
Modifying the UserId	158
Modifying Password	158
Modifying defaultObjectSecurityPolicy	159
Modifying Privileges.	161
Modifying SymbolLists	162
Disable and Enable User Logins	163
Disable and Enable Commits by User	165
8.5 Configuring GemStone Authentication	166
Configuring GemStone Login Security.	166
Limiting Choice of Passwords	167
Disallowing Particular Passwords	169
Disallowing Reuse of Passwords	169
Password Aging – Require Periodic Password Changes.	170
Account Aging – Disable Inactive Accounts.	171
Enabling Account Aging and lastLoginTime	172
Limit Logins Until Password Is Changed	173
Limit Concurrent Sessions by a Particular UserId	173
Limit Login Failures	174
8.6 Configuring UNIX Authentication.	174
8.7 Configuring LDAP Authentication	175
LDAP authentication without anonymous binds	176
Validating passwords	177
8.8 Configure SingleSignOn Authentication	178
Kerberos concepts	178
KerberosPrincipal and AllKerberosPrincipals.	179
Setting up Kerberos Authentication in GemStone	179
Using Groups to authenticate with Kerberos	180
KerberosPrincipal available to all users	181

8.9 Tracking User Logins	182
Login logging	182
Login Hook	182
Debugging	183
Chapter 9. Managing Repository Space	185
9.1 The Repository and Extents	185
Repository Growth	186
How To Check Free Space	186
9.2 Adding and Removing Extents	187
To Add an Extent While the Stone is Running	188
Possible Effects on Other Sessions	188
Repository>>createExtent:	188
Repository>>createExtent:withMaxSize:	189
To Remove an Extent	189
9.3 Reallocating Existing Objects Among Extents	190
To Reallocate Objects Among a Different Number of Extents	190
To Reallocate Objects Among the Same Number of Extents	190
9.4 Shrinking the Repository.	191
9.5 Checking Page Fragmentation.	194
9.6 Disk Space and Commit Record Backlogs	194
Handling signals indicating a commit record backlog	195
9.7 Recovering from Disk-Full Conditions	196
Repository Full	196
Chapter 10. Managing Transaction Logs	199
10.1 Overview	199
Logging Modes	201
Restoring Transactions to a Backup	202
How the Logs Are Used.	202
10.2 How To Manage Full Logging	203
To Archive Logs	204
To Add a Log at Run Time	205
To Force a New Transaction Log	206
To Initiate a Checkpoint	206
To Change to Partial Logging	207
10.3 How To Manage Partial Logging	207
To Change to Full Logging.	207
10.4 How To Recover from Tranlog-Full Conditions	208
Transaction Log Space Full.	208

Chapter 11. Making and Restoring Backups **209**

11.1 Overview	209
Warm and Hot Standbys	210
Version Compatibility	210
11.2 Types of Backups	211
11.3 How To Make an Extent Snapshot Backup	212
Extent Snapshot Backup when the Repository is shutdown.	212
Extent Snapshot Backup when the Repository is running	213
11.4 How To Make a Smalltalk Full Backup.	215
Monitoring and Performance	217
Compressed Backups	218
Verifying a Backup is Readable.	218
Checking Backup Start and Completion	219
11.5 How to Restore from Backup	219
Restoring from an Extent Snapshot Backup	220
Restoring from a Full Backup	222
Steps in restoring a full backup	223
11.6 How to Make and Restore Secure Backup	225
Creating a secure backup	226
Setting the certificate directories	226
Create an unencrypted secure backup	227
Create an encrypted backup.	228
Restoring a secure backup.	228
Restore an unencrypted backup.	229
Restore an encrypted backup	229
Verifying the digital signature	229
11.7 How to Restore Transaction Logs.	230
11.8 Special Cases and Errors in Restore.	232
Precautions When Restoring a Subset of Transaction Logs	232
Restoring Logs up to a Specific Log	234
Restoring Logs to a Point in Time	235
Errors While Restoring Transaction Logs	236
Missing Transaction Log File	236
Truncated or Corrupt Transaction Log File	236
Recovering from File System Problems	237

Chapter 12. Warm and Hot Standbys **239**

12.1 Overview	239
12.2 Warm Standby	240
Setup and run the warm standby.	240
Activate the warm standby in case of failure in the primary	241
12.3 Hot Standby	242
Hot standby processes	242
logsender.	242

logreceiver	243
Continuous Restore Mode	243
Transaction Record Transmittal	243
Multiple standby repositories	244
To setup and run the hot standby	244
Activate the hot standby in case of failure in the primary	245
Planned failovers	246
Connecting using SSL Mode	246
12.4 Tuning a Warm or Hot Standby	247
Tuning Reclaim	248
Chapter 13. Managing Memory	249
13.1 Memory Organization	249
13.2 Configuring Temporary Memory Usage	250
Configuration Options	250
Methods for Computing Temporary Object Space	251
Debugging out-of-memory errors	252
Recording Out of Memory Information to CSV file	254
Signal on low memory condition	254
Chapter 14. Managing Growth	255
14.1 Basic Concepts	255
Shadow or Dead?	257
What Happens to Garbage?	260
Admin and Reclaim Gems	261
Admin and Reclaim Gem configuration parameters	261
GemStone's Garbage Collection Mechanisms	262
Marking	262
Reclaiming	263
GcLock	263
Symbol Garbage Collection	263
14.2 MarkForCollection	264
Impact on Other Sessions.	265
Scheduling markForCollection	265
14.3 Epoch Garbage Collection	266
Running Epoch Garbage Collection.	266
Tuning Epoch	267
Cache Statistics	271
14.4 Reclaim	272
Tuning Reclaim	273
Reclaim Configuration Parameters	273
Reclaim Commit Frequency	274
Controlling the impact of reclaim	274

Speeding up reclaim274
Avoiding disk space issues275
Cache Statistics275
14.5 Running Admin and Reclaim Gems275
Configuring Admin Gem276
Configuring Reclaim Gem276
Starting GcGems276
Stopping GcGems277
Adjusting the number of Reclaim sessions277
14.6 Further Tuning Garbage Collection.278
Multi-Threaded Scan.278
Tuning Multi-Threaded Scan278
Memory Impact279
Identifying Sessions Holding Up Voting280
Tuning Write Set Union Sweep280
Identifying Sessions Holding Up Page Reclaim.280
Finding large objects that are using excessive space281
Identify Larger Objects in the Repository281
Finding named objects that are large281
Finding References to an Object that prevent garbage collection282
Full reference path282
All References284

Appendix A. GemStone Configuration Options 285

A.1 How GemStone Uses Configuration Files285
System Configuration File286
Executable Configuration File.287
Creating or Using a System Configuration File288
Creating an Executable Configuration File288
Nesting Configuration Files289
Naming Executable Configuration Files289
Naming Conventions for Configuration Options.291
A.2 Configuration File Syntax291
Errors in Configuration Files292
Syntax Errors292
Option Value Errors293
Alternate ways to specify configuration parameter values293
A.3 Configuration Options294
CONFIG_WARNINGS_FATAL294
DBF_ALLOCATION_MODE294
DBF_EXTENT_NAMES294
DBF_EXTENT_SIZES295
DBF_PRE_GROW295
DBF_SCRATCH_DIR296
DUMP_OPTIONS296

GEM_ABORT_MAX_CRIS	296
GEM_CACHE_WARMER_ARGS	296
GEM_CACHE_WARMER_MID_CACHE_ARGS	297
GEM_COMPRESS_TRANLOG_RECORDS	297
GEM_FREE_FRAME_CACHE_SIZE	297
GEM_FREE_FRAME_LIMIT	297
GEM_FREE_PAGEIDS_CACHE	298
GEM_HALT_ON_ERROR	298
GEM_KEEP_MIN_SOFTREFS	298
GEM_KERBEROS_KEYTAB_FILE	299
GEM_KEYRING_DIRS	299
GEM_MAX_SMALLTALK_STACK_DEPTH	299
GEM_NATIVE_CODE_ENABLED	299
GEM_PGSVR_COMPRESS_PAGE_TRANSFERS	300
GEM_PGSVR_FREE_FRAME_CACHE_SIZE	300
GEM_PGSVR_FREE_FRAME_LIMIT	300
GEM_PGSVR_UPDATE_CACHE_ON_READ	301
GEM_PGSVR_USE_SSL	301
GEM_PRIVATE_PAGE_CACHE_KB	301
GEM_READ_AUTH_ERR_STUBS	301
GEM_REPOSITORY_IN_MEMORY	302
GEM_RPCGCI_TIMEOUT	302
GEM_RPC_KEEPALIVE_INTERVAL	302
GEM_RPC_USE_SSL	302
GEM_STATMONITOR_ARGS	303
GEM_STATMONITOR_MID_CACHE_ARGS	303
GEM_SOFTREF_CLEANUP_PERCENT_MEM	303
GEM_TEMPOBJ_AGGRESSIVE_STUBBING	303
GEM_TEMPOBJ_CACHE_SIZE	304
GEM_TEMPOBJ_CONSECUTIVE_MARKSWEEP_LIMIT	304
GEM_TEMPOBJ_MESPACE_SIZE	305
GEM_TEMPOBJ_OOMSTATS_CSV	305
GEM_TEMPOBJ_OOPMAP_SIZE	305
GEM_TEMPOBJ_POMGEN_PRUNE_ON_VOTE	306
GEM_TEMPOBJ_POMGEN_SCAVENGE_INTERVAL	306
GEM_TEMPOBJ_POMGEN_SIZE	306
GEM_TEMPOBJ_SCOPES_SIZE	306
GEM_TEMPOBJ_START_ADDR	307
INCLUDE	307
KEYFILE	307
LOG_WARNINGS	307
SHR_NUM_FREE_FRAME_SERVERS	307
SHR_PAGE_CACHE_LARGE_MEMORY_PAGE_POLICY	308
SHR_PAGE_CACHE_LARGE_MEMORY_PAGE_SIZE_MB	308
SHR_PAGE_CACHE_LOCKED	309
SHR_PAGE_CACHE_NUM_PROCS	309
SHR_PAGE_CACHE_NUM_SHARED_COUNTERS	309

SHR_PAGE_CACHE_PERMISSIONS	310
SHR_PAGE_CACHE_SIZE_KB.	310
SHR_SPIN_LOCK_COUNT.	310
SHR_TARGET_FREE_FRAME_COUNT	311
SHR_WELL_KNOWN_PORT_NUMBER	311
STN_ADMIN_GC_SESSION_ENABLED	311
STN_ALLOCATE_HIGH_OOPS	311
STN_ALLOW_NFS_EXTENTS	312
STN_ALLOW_NO_SESSION_INIT	312
STN_CACHE_WARMER	312
STN_CACHE_WARMER_ARGS	312
STN_CACHE_WARMER_SESSIONS	313
STN_CACHE_WARMER_WAIT_MODE	313
STN_CHECKPOINT_INTERVAL	313
STN_COMMIT_QUEUE_THRESHOLD.	313
STN_COMMIT_RECORD_BM_CACHING.	314
STN_COMMIT_RECORD_QUEUE_SIZE	314
STN_COMMIT_TOKEN_TIMEOUT.	314
STN_COMMITS_ASYNC	314
STN_CR_BACKLOG_THRESHOLD.	314
STN_DISABLE_LOGIN_FAILURE_LIMIT	315
STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT	315
STN_DISKFULL_TERMINATION_INTERVAL	315
STN_EPOCH_GC_ENABLED	316
STN_EXTENT_IO_FLAGS	316
STN_FREE_FRAME_CACHE_SIZE	316
STN_FREE_SPACE_THRESHOLD.	316
STN_GEM_ABORT_TIMEOUT.	317
STN_GEM_LOSTOT_TIMEOUT	317
STN_GEM_PGSVR_CONNECT_TIMEOUT	317
STN_GEM_PRIVATE_PGSVR_ENABLED	318
STN_GEM_TIMEOUT	318
STN_GROUP_COMMITS	318
STN_HALT_ON_FATAL_ERR.	318
STN_LISTENING_ADDRESSES	318
STN_LOGIN_LOG_DIR	319
STN_LOGIN_LOG_ENABLED.	319
STN_LOGIN_LOG_HALT_ON_ERROR	320
STN_LOGIN_LOG_MAX_SIZE.	320
STN_LOG_IO_FLAGS	320
STN_LOG_LOGIN_FAILURE_LIMIT	321
STN_LOG_LOGIN_FAILURE_TIME_LIMIT	321
STN_LOOP_NO_WORK_THRESHOLD	321
STN_MAX_AIO_RATE	322
STN_MAX_AIO_REQUESTS	322
STN_MAX_GC_RECLAIM_SESSIONS	322
STN_MAX_LOGIN_LOCK_SPIN_COUNT	322

STN_MAX_REMOTE_CACHES	323
STN_MAX_SESSIONS	323
STN_MAX_VOTING_SESSIONS	323
STN_NUM_AIO_WRITE_THREADS	323
STN_NUM_GC_RECLAIM_SESSIONS	324
STN_NUM_LOCAL_AIO_SERVERS	324
STN_OBJ_LOCK_TIMEOUT	324
STN_PAGE_MGR_COMPRESSION_ENABLED	324
STN_PAGE_MGR_MAX_WAIT_TIME	325
STN_PAGE_MGR_PRINT_TIMEOUT_THRESHOLD	325
STN_PAGE_MGR_REMOVE_MAX_PAGES	325
STN_PAGE_MGR_REMOVE_MIN_PAGES	325
STN_PGSVR_PORT_RANGE	326
STN_PRIVATE_PAGE_CACHE_KB	326
STN_REMOTE_CACHE_PGSVR_TIMEOUT	326
STN_REMOTE_CACHE_STARTUP_TIMEOUT	326
STN_REMOTE_CACHE_TIMEOUT	327
STN_SHR_TARGET_PERCENT_DIRTY	327
STN_SIGNAL_ABORT_AGGRESSIVE	327
STN_SIGNAL_ABORT_CR_BACKLOG	327
STN_SMC_SPIN_LOCK_COUNT	328
STN_STATMONITOR_ARGS	328
STN_STONE_CACHE_STARTUP_TIMEOUT	328
STN_SYMBOL_GC_ENABLED	329
STN_SYMBOL_GEM_TEMPOBJ_CACHE_SIZE	329
STN_TRAN_FULL_LOGGING	329
STN_TRAN_LOG_DEBUG_LEVEL	330
STN_TRAN_LOG_DIRECTORIES	330
STN_TRAN_LOG_LIMIT	330
STN_TRAN_LOG_PREFIX	330
STN_TRAN_LOG_SIZES	331
STN_TRANQ_TO_RUNQ_THRESHOLD	331
STN_WELL_KNOWN_PORT_NUMBER	331
A.4 Runtime-only Configuration Options	332
GemConvertArrayBuilder	332
GemDropCommittedExportedObjs	332
GemExceptionSignalCapturesStack	332
LogOriginTime	332
SessionInBackup	332
StnCurrentTranLogDirId	333
StnCurrentTranLogNames	333
StnLogFileName	333
StnLogGemErrors	333
StnLoginsSuspended	333
StnMaxReposSize	333
StnMaxSessions	333
StnStandbyRole	333

StnSunsetDate.334
StnTranLogOriginTime334

Appendix B. GemStone Utility Commands 335

copydbf336
Compression of copydbf output338
Using copydbf to access information on a file.338
copydbf with remote nodes339
copydbf with raw partitions.340
gslis t341
gslis t on Windows343
pageaudit344
pstack345
removedbf346
startcachewarmer347
startlogreceiver349
startlogsender351
startnetldi353
netldid354
startstone355
stoned356
statmonitor357
stoplogreceiver360
stoplogsender361
stopnetldi362
stopstone363
topaz364
vsd366
waitstone367

Appendix C. Network Resource Strings (NRS) 369

A.1 Using NRS369
GsNetworkResourceString370
GEMSTONE_NRS_ALL370
Controlling log file names and locations.371
A.2 NRS Syntax.373
Spaces and Special Characters.373
Syntax373

Appendix D. GemStone Kernel Objects 377

Non-Numeric Constants377
Numeric Constants377
Repository and GsObjectSecurityPolicies378

Global Variables and Collections	379
Current TimeZone	383
Zoneinfo	384
Utilities	385
<i>Appendix E. Environment Variables</i>	387
Public Environment Variables.	387
System Variables Used by GemStone	391
Reserved Environment Variables	392
<i>Appendix F. Object State Change Tracking</i>	393
A.1 Overview	393
A.2 Tranlog Analysis Scripts.	394
Script Prerequisites	394
Output	394
Tranlog Assumptions.	394
Filter Criteria.	395
printlogs	396
Examples	396
searchlogs	397
Examples	397
A.3 Tranlog Structure.	397
Tranlog Entries	398
Tranlog Entry Types	399
Very Large Objects	400
Full vs. Normal Mode.	400
A.4 Example of Tranlog Analysis	402
Tracking Changes to an Employee	403
Changed vs. new objects	404
Details of Changes to an Employee	405
A.5 Further Analysis	407
Class Operations.	407
Deleted Objects	407
Managing Volume.	407

Administration of the GemStone/S Environment

This manual provides information that is useful for configuring and administering a GemStone/S 64 Bit™ installation. This includes configuring GemStone installations, both new systems and systems that have grown or whose requirements have changed; how to keep a GemStone system running smoothly, including recovering from problems and troubleshooting; and information helpful in tuning each of the subsystems that allow GemStone to operate efficiently and seamlessly.

This chapter provides some basic information about the parts of a GemStone installation, and an overview of the Administration process.

The details of installing GemStone software, and upgrading from earlier versions, is described in detail in the *Installation Guide*, which are platform-specific. Refer to these guides for information on tuning your OS parameters for most efficient use of GemStone.

Basic GemStone/S 64 Bit Architecture (page 24)

the components of the GemStone/S 64 Bit server and clients.

Starting GemStone and Logging In (page 25)

the steps of starting a GemStone server and logging in a client

Authentication and Authorization (page 29)

an overview on how GemStone manages authentication and authorization

Transactions and commit records (page 30)

how GemStone manages data between multiple users.

Files and Directories (page 31)

the disk files that the GemStone server requires and creates.

Options for Configuring (page 33)

configuration files, environment variables and other ways to specify configuration parameters.

Avoiding risk of Data loss (page 36)

information on GemStone's features to ensure against loss of critical data

Running a Second Repository (page 38)

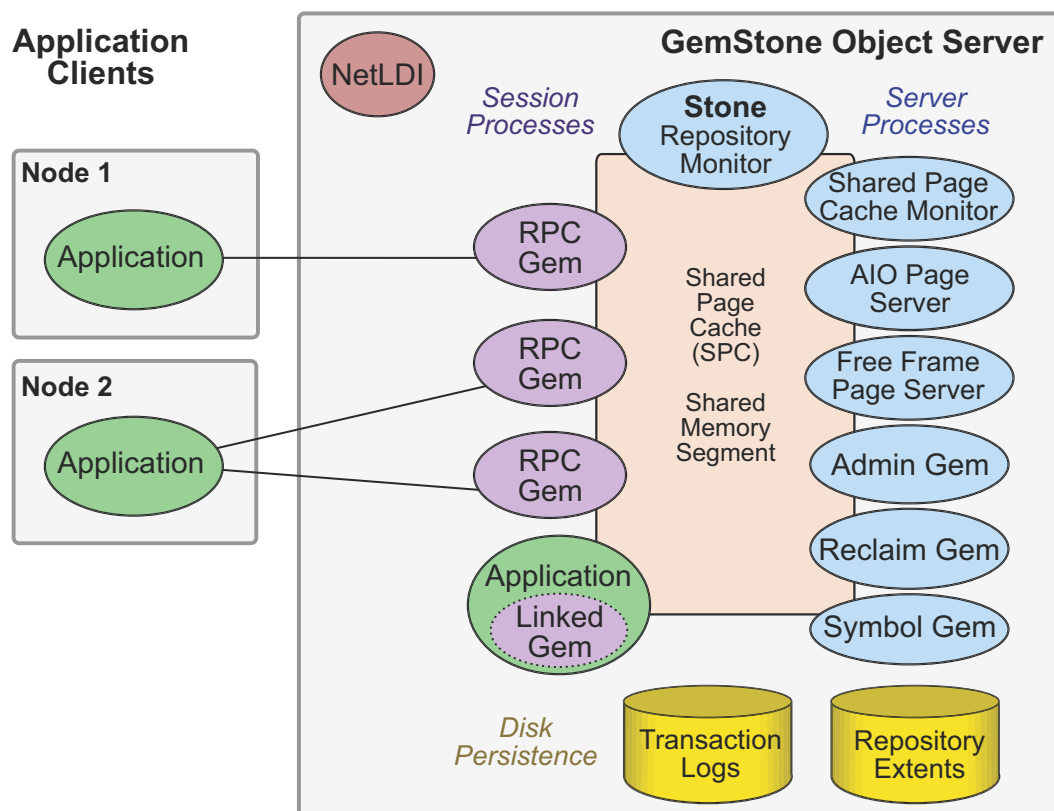
instructions on running two repositories on the same machine.

1.1 Basic GemStone/S 64 Bit Architecture

Figure 1.1 shows the basic GemStone/S 64 Bit architecture. The GemStone object server can be thought of as having two parts. The *server processes* consist of the Stone repository monitor and a set of subordinate processes. These processes provide resources to individual Gem *session processes*, which are servers for application clients.

While in this simple configuration, the remote nodes hold only the Application Clients, there are further options to distribute components over multiple nodes, described later in this manual.

Figure 1.1 The GemStone Object Server



Keys parts that define the server configuration include:

Server Processes

- ▶ The *Stone repository monitor* process acts as a resource coordinator. It synchronizes critical repository activities and ensures repository consistency.
- ▶ The *shared page cache (SPC)* is a shared memory segment that holds the pages on which data and meta-information are stored. Pages are read into *frames* in the cache. Pages are read from and written to the extent files, and used for temporary data. A larger cache provides better performance since it reduces disk I/O; the cache may be large enough to hold all data in the extents.

- ▶ The *shared page cache monitor* process creates and maintains the shared page cache. The monitor balances page allocation among processes, ensuring that a few users or large objects do not monopolize the cache.
- ▶ The *AIO page server* performs asynchronous I/O, in particular to update the extent files based on changes in the shared page cache.
- ▶ The *Free Frame Page Server* helps ensure that there are free frames in the shared page cache so they are available for the Gems to use.
- ▶ The *Admin Gem* performs administrative garbage collection tasks.
- ▶ The *Reclaim Gem* performs reclaim, cleaning up old versions of objects and dead objects, so that the pages can be reused.
- ▶ The *Symbol Gem* is responsible for creating all new Symbols, based on session requests that are managed by the Stone.

Disk-based Persistence

- ▶ Objects are stored on disk in one or more *extents*, which can be files in the file system, data in raw partitions, or a mixture.
- ▶ *Transaction logs* permit recovery of committed data if a system crash occurs, and in *full logging mode* allows transaction logs to be used with GemStone backups for full recovery of committed transactions in case of disk failure.

Session Processes

- ▶ *Gem* sessions can be Gem processes in an RPC (Remote Procedure Call) login, or bound with the client application using shared libraries in a linked login.

Communications process

- ▶ The NetLDI (Network Long Distance Information) listens on a configured port for connections, to establish logins and perform other tasks in distributed systems.

1.2 Starting GemStone and Logging In

Stone

A configured GemStone server is started by the **startstone** command, which starts the Stone repository monitor. The Stone in turn starts the shared page cache monitor and other server processes. The extent files are attached, and a transaction log opened for writing.

The Stone is named; by default, *gs64stone*. Only one Stone with a given name can run on a particular node.

You may run more than one Stone on a node, as long as names are different and the extent files and transaction logs are in different locations or have different names.

Configuring the Stone and other server processes is described in Chapter 2, “Configuring the GemStone Server”, starting on page 39.

For details on starting the Stone and other server processes, and troubleshooting issues, see “Starting the GemStone Server” on page 103.

Shared Page Cache

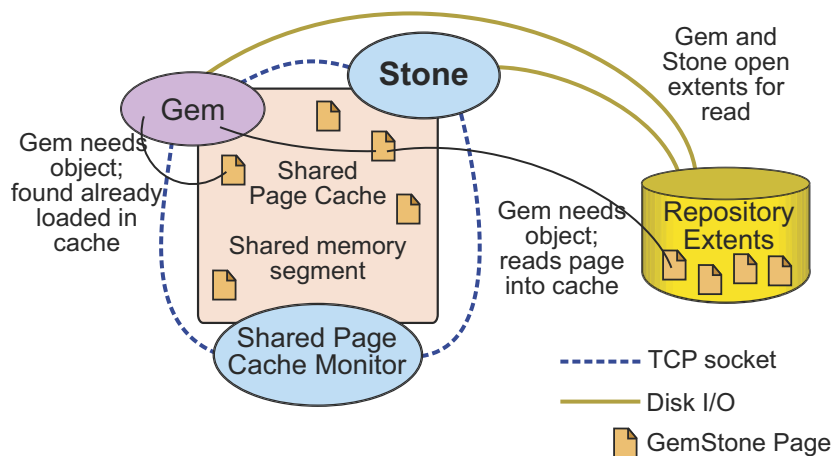
The GemStone shared page cache system has two parts: the shared page cache itself, a shared memory segment; and a monitor process, the shared page cache monitor (`shrpcmonitor`). Figure 1.2 shows the connections between these and other components on a single node.

The shared page cache resides in a segment of the operating system's virtual memory that is available to any authorized process. When the Stone or a Gem session process needs to access an object in the repository, it first checks to see whether the page containing that object is already in the cache. If the page is already present, the process reads the object directly from shared memory. If the page is not present, the process reads the page from the disk into the cache, where all of its objects also become available to other processes.

The shared page cache monitor also has a name, which is derived from the name of the Stone repository monitor and the *Host Identifier*; for instance, `gs64stone~d7e2174792b1f787`.

Each Stone has a single shared page cache on its own node, and may have remote page caches on other nodes in distributed configurations (discussed in detail in Chapter 5). The Stone spawns the shared page cache monitor automatically during startup, and the shared page cache monitor creates the shared memory region and allocates the semaphores for the system. All sessions that log into this Stone connect to this shared page cache and monitor process.

Figure 1.2 Pages in the Shared Page Cache



Shared Cache Size

GemStone performance is almost always improved by making the Shared Page Cache larger, up to a size in which the entire repository can be held in memory.

With a very large cache, it is recommended to run cache warming on startup, so that at least the pages containing the object table (used for lookup) are loaded into the cache. Otherwise, the first access to data will require the pages to be read from disk.

With very large caches on Linux and AIX, you can improve memory use by configuring the cache to use large memory pages; 16MB on AIX, and 2MB and 1GB pages on Linux.

OS memory and Swapping

As described in the Installation Guide, you are likely to need to configure your OS to support a sufficiently large shared memory segment.

Be careful not to make the shared page cache so large that it forces swapping. You should ensure that your system has sufficient RAM to hold the configured shared page cache, with extra space for the other memory requirements.

For details on configuring the shared page cache, see “Shared Page Cache” on page 41.

NetLDI

A NetLDI also must be configured and explicitly started. While there are some cases, in which there are only linked logins, that do not require a NetLDI, most installations will need to start a NetLDI.

Each NetLDI has a name, but may also be referred to by its listening port number. The default name is *gs64ldi*; by using this name and setting this up in the services database on all your nodes, you can skip the step of specifying the NetLDI.

You may run more than one NetLDI on a node, related to the same GemStone system or an entirely different system, as long as the names and ports are different.

If you are using more than one NetLDI on a particular node, and one of them is running with the default name, use particular care to ensure that all logins contact the correct NetLDI.

Setting up the NetLDI is described in Chapter 4, “NetLDI and Interprocess Access”, starting on page 71.

For details on starting the NetLDI see “Starting a NetLDI” on page 109.

Logging in Gem Sessions

Client applications include:

- ▶ the topaz command line tool provided as part of the server
- ▶ client Smalltalk applications such as VisualWorks and VA Smalltalk using GBS
- ▶ web applications
- ▶ other custom GCI application

Client application are configured to load the GCI shared library, provided as a .dll, .dynlib, or .so file, containing the Gem API.

The login commands include, of course, GemStone authorization details. The specification of the Stone and node must be provided, and the location where the Gem process is to be run. The details need to include the name or port of the NetLDI that sets up the login. This information is provided using special *NRS* (*Network Resource String*) syntax.

The NetLDI uses that information to fork the Gem process, trigger the startup of any other required server processes, and setup the interprocess communications.

Linked vs. RPC

In a linked login, GCI shared libraries are loaded into the client application (linked topaz, GBS, or another), and Gem is part of that client application. Since the Gem is part of the server, this is only possible on platforms that support the GemStone/S 64 Bit server, i.e., not on Windows.

In an RPC login, the Gem is a separate process from the client application. The two processes may be on the same node or on different nodes, and communicate via remote procedure calls.

Gemservers for Large Configurations

Each Gem is an individual process. Depending on your hardware, there is a limit to the number of processes that you can run before degrading performance. For very large configurations, it may be useful to establish a separate node specifically to run Gem sessions, with a high bandwidth connection between the repository server and the Gem server. The Gems on this gemserver have the performance benefit of a remote shared page cache on this node.

Configuring client Gems is described in Chapter 3, “Configuring Gem Session Processes”, starting on page 63.

For more information on logging in a Gem session, see “Starting a GemStone Session” on page 110.

NRS (Network Resource String)

GemStone’s proprietary NRS syntax is used to specify the name and location of each part of the GemStone system. These are widely used in utility commands and in login parameters, to specify the location and name of the Stone and NetLDI, and where the other GemStone processes should run.

Appendix C, “Network Resource Strings (NRS)” describes NRS strings in detail.

The most commonly used syntax, as in the examples in this manual, has the form:

`!@nodeNameOrId#netldi:netldiNameOrPort!stoneOrGemService`

- ▶ The “@nodeNameOrId” can be omitted for the local node.
- ▶ the “#netldi:netldiNameOrPort” can be omitted if the NetLDI is running with the default name `gs64ldi`, and the name and port are configured in the services database.
- ▶ `stoneOrGemService` is the name of the running Stone, or `gemnetobject` or another Gem service. If you omit both node and netldi, you do not need the ! dividers; you can use only `stoneOrGemService`.

1.3 Authentication and Authorization

GemStone provides several levels of security; UNIX user accounts must have permission and may require authentication; login to GemStone requires a GemStone account and authentication; and objects and access to data and important functions is controlled within GemStone using SecurityPolicies and Privileges. In full logging mode, all commits to the repository are recorded in the transaction logs, which can be analyzed to determine the source of changes; and the system can be configured to log all logins.

GemStone UserIds and login

Logins to GemStone are done as GemStone users, which are instances of UserProfile. UserProfiles have names and passwords that are unrelated to UNIX userIds, although GemStone UserProfiles can be created that correspond to the UNIX account names.

Login authentication can be configured to use UNIX or LDAP, in which case either the names must match, or the UserProfile configured with the mapping. UserProfiles can also be configured with single sign-on using Kerberos, by setting up the appropriate mapping within GemStone.

There are several built-in system accounts, including SystemUser, which is similar to a “root” user, used for upgrades; and DataCurator, the administrative user for tasks such as backups and User administration. Other system accounts are used for garbage collection and symbol creation.

When GemStone UserProfile authentication is controlled by GemStone, there are a number of ways to restrict password choice, require password changes, and disable accounts in cases such as too many failed logins. Internally, passwords are stored only in the encrypted form and there is no facility to decrypt them.

A number of important operation, such as code modification and garbage collection, may be restricted to certain users by using privileges. UserProfile may be granted privileges that allow them to perform these operations.

Configuring and Administering Users, Groups, and Privileges is described in Chapter 8, “User Accounts and Security”.

Authorization to Access Data

Within GemStone, Objects are associated with Security Policies, that allow specific GemStone Users or Groups of users to write, read, or have no access to particular objects. This provides a highly granular way of protecting critical data.

How to apply Security Policies to your data is described in the *GemStone/S 64 Bit Programming Guide*.

File access and authorization

As a multiuser system, GemStone must allow applications running under different UNIX userIds to create processes that access the repository disk files, while ensuring that the disk files are protected against unauthorized access or modification, both intentional and accidental. This involves managing the permissions for important files and controlling the process owners and groups. Configuring security is described in Chapter 4.

Initial login requests to GemStone connect using SSL (Secure Socket Layer; more specifically, TLS protocol using OpenSSL). For processes on the same machine, communications after the login continue using primarily shared memory. Remote socket connections may be SSL or not, depending on your security requirements.

Setting up file permissions and authentication is described in Chapter 4.

1.4 Transactions and commit records

GemStone maintains a consistent view for each user, and all changes that are made persistent and visible to other users are done within transactions. Transactions are committed to make changes persistent; sessions abort to discard modifications and update the objects in their view with changes made by other users.

For more information on transactions and commits, refer to the *GemStone/S 64 Bit Programming Guide*.

Object views

On login, each GemStone user acquires a “view” of the objects in the repository. This view is maintained as long as you are using it, even if the objects have been modified by other users. You may make modifications to the objects in your view, or create new objects, but these changes are transient unless and until you commit, in which case they become persistent and can be viewed by other users.

Making new objects persistent also requires that they be connected to an existing persistent object. Objects that are not reachable by other objects are subject to garbage collection.

If another user has made a conflicting change, your commit may fail, and you can get a report of the conflicts. You may need to discard the changes. This can be avoided by locking objects prior to modifying them; reduced-conflict classes allows certain kinds of conflict to be automatically resolved.

Session transactional state

Each Gem session is always either “in transaction” or not in transaction (a third state, transactionless, is normally only used for background tasks). Sessions that are in transaction may commit changes; when not in transaction, data may be viewed and objects modified, but these modifications are transient. To make persistent changes, you must begin a transaction, make the changes, and commit successfully.

Sessions that are in a transaction can commit, and any session can abort. When a session commits or aborts, its view is updated with to the most recent view, including any changes by other users.

Commit records

GemStone is designed to accommodate large numbers of users. Each user may have his or her own view of the repository; and each of these views must be maintained as long as they are in use. GemStone maintains these views as Commit Records.

Over time, each of these sessions will commit changes (or abort), at which point the view is no longer required. However, since commit records are sets of changes from a previous view, GemStone must maintain the commit record of the session that has been logged in without commit or abort for the longest time (the oldest commit record) and every intermediate commit record up to the current one.

Commit records are stored in the repository, and a large commit record backlog can use a large amount of repository space. GemStone can signal sessions that are causing a backlog, but applications in multiuser system should be designed to abort or commit and respond to signals to avoid creating backlogs. A long-lasting commit record backlog can fill up all disk space such that the GemStone cannot avoid shutting down.

1.5 Files and Directories

GemStone Installation

The GemStone installation process, as described in the *Installation Guide* for your platform, describes both how to configure your operating system, and the details of installing GemStone.

After installation, you should have a directory containing the executables, shared libraries, extents and other required or useful files. Not all are required, and they do not need to be located in this shared directory structure.

This directory is referenced by the required environment variable `$GEMSTONE`.

GemStone shared libraries

Your GemStone installation includes shared library files as well as executables. Access to these shared library files is required for the GemStone executables. In the standard installation of the GemStone software, these shared libraries are located in the `$GEMSTONE/lib` and `$GEMSTONE/lib32` directories.

For installations that do not include a full server, such as remote nodes that are only running client applications, these libraries may be put in a directory other than this standard. See the *Installation Guide* for more information.

Separate *GemStone/S 64 Bit Installation Guides* are provided for each supported platform, containing OS configuration, installation, and upgrade information. For Windows clients, see the *GemStone/S 64 Windows Client*.

Lock file directory

In addition to the normal installation directory, GemStone requires access to two directories under `/opt/gemstone/`:

- ▶ `/opt/gemstone/locks` is used for lock files, which among other things provide the names, ports, and other important data used in interprocess communication and reported by **gslist**.

Under normal circumstances, you should never have to directly access files in this directory. To clear out lock files of processes that exited abnormally, use **gslist -c**.

- ▶ `/opt/gemstone/log` is the default location for NetLDI log files, if `startnetldi` does not explicitly specify a location using the `-l` option.

If `/opt/gemstone/` does not exist, GemStone may use `/usr/gemstone/` instead.

Alternatively, you can use the environment variable `GEMSTONE_GLOBAL_DIR` to specify a different location. Since the files in this location control visibility of GemStone processes to one another, all GemStone processes that interact must use the same directory.

Host Identifier

`/opt/gemstone/locks` (or an alternate directory, as described above) is also the location for a file named `gemstone.hostid`, which contains the unique host identifier for this host. This file is created by the first GemStone process on that host to require a unique identifier, by reading eight bytes from `/dev/random`. This unique `hostid` is used instead of host name or IP address for GemStone inter-process communication, avoiding issues with multi-homed hosts and changing IP address.

You can access the host identifier for the machine hosting the gem session using the method `System class >> hostId`.

Extents, Tranlogs, and disk space

GemStone data is preserved on disk in one or more extent files. These extents include the classes and objects that make up GemStone, as well as the classes and objects that make up your application. When GemStone is shutdown, these extents provide the complete set of objects that compose the application.

Each time a session commits changes to the repository, the changes are made on pages in the shared page cache, which are eventually written to disk by the AIO page server. A commit also writes a record in the current transaction log, which is immediately written to disk.

While GemStone is running, it periodically makes a checkpoint, at which point all dirty pages in the shared page cache are written to disk, and the root page is updated to a new consistent state. GemStone also writes a checkpoint on an orderly shutdown, and before a backup.

Between checkpoints, updates to the repository are recorded in the transaction logs. If GemStone shuts down unexpectedly, it will startup at the point of the most recent checkpoint in the repository. Changes since that checkpoint are restored from the transaction logs.

Disk Usage for Extents and Transaction Logs

On a single disk, multiple processes writing to both extents and transaction logs will encounter contention, since only one physical write to disk media at a time is possible.

For this reason, if you are not using a SAN or a RAID device that avoids single disk contention, using multiple physical disk drives is recommended for best performance.

Disk configuration

Extents benefit from efficiency of both random access (16 KB repository pages) and sequential access. Sequential access is important for such operations as garbage collection and making or restoring backups, while random access is used for most data I/O. Using

RAID devices or striped file systems that cannot efficiently support both random and sequential access may reduce overall performance.

Transaction logs use sequential access exclusively, so the devices can be optimized for that access.

Raw Partitions

Each raw disk partition is like a single large sequential file, with one extent or one transaction log per partition.

In general, placing extents on file systems is as fast as using raw partitions, but transaction logs on raw partitions is likely to yield better performance in an update-intensive application.

Process Log Files

Each GemStone process creates or appends to a log file. These log files include process details, startup configuration information, and messages about any errors that occur. The location and file name of most processes is configurable.

When the process exits, by default some processes leave their log files in place for possible later diagnostic use. Other process types delete log files on a clean exit. A process that exits with an error never deletes its log files.

Note that linked sessions, which do not have an independent Gem process, do not create log files. Log file output is sent to stdout for the client application.

System Clock and GemStone times

The system clock should be set to the correct time. When GemStone opens the repository at startup, it compares the current system time with the recorded checkpoint times as part of a consistency check. A system time earlier than the time at which the last checkpoint was written may be taken as an indication of corrupted data and prevent GemStone from starting.

TimeZones

Internally GemStone uses times in GMT. It is not necessary to adjust GemStone for changes to and from daylight savings time.

GemStone uses an internal TimeZone setting to adjust times for display in local time. This uses the Olson Tz database.

1.6 Options for Configuring

GemStone is a very flexible system, with many options for configuring details. Almost all of these have defaults; for an “average” system, you will only need to set a few basics: the extent files, transaction logs, and the size of the shared cache.

However, few systems are average. Your application may have larger or smaller amounts of data relative to the number of users, or have a higher commit rate, or a commit rate that varies widely over the course of a day; or your system have particular requirements for reliability, security, or performance; or your system may be distributed over multiple

nodes. For example, a nightly batch data archiving process may have very different requirements than a monitoring process that performs frequent small commits.

An application's configuration is set by assigning values in one or more configuration files. These can be modified, or work in concert with, specific environment variables, arguments to the command-line GemStone utilities, and by executing Smalltalk code that adjusts configuration values or behavior at run-time.

This flexibility permits assigning system-wide values that include both fixed and variable configuration settings, while also allowing various components to override these settings when they have specific needs.

Configuration files

GemStone uses configuration files to hold the specifications for most important configuration details. This includes the names and locations of the extent files, the sizes of the various caches, and many parameters designed for tuning. While there are many parameters, only a few of them are required; in most cases the default is sufficient.

On a new Gemstone system, you must determine the basic configuration before starting. The details on how to establish the configuration are described in Chapter 2, "Configuring the GemStone Server" for the server and Chapter 3, "Configuring Gem Session Processes" for Gems.

GemStone distinguishes between the system configuration file, which applies to the Stone and other system processes, and application configuration files, which apply for Gem sessions (linked and RPC). If all GemStone processes are on a single node, the parameters can all be in a single configuration file. Multiple files can be setup, which apply to different tasks or reside on different nodes.

How configuration files are used, and an alphabetic list of parameters, is provided in Appendix A. Specific parameters are discussed throughout this manual.

Environment variables

There are a number of environment variables that GemStone uses. The most important of these is \$GEMSTONE, which is required, and indicates the directory in which GemStone is installed.

Other environment variables provide file locations for configuration files themselves, and other information that is needed before a process is able to read the configuration file.

Finally, some logging and debugging settings are provided as environment variables to allow debugging without affecting other sessions in a multiuser system.

Environment variables are listed in Appendix E. Their use is described in various sections to which they apply.

Utility command arguments

When GemStone is started up, the startup tasks are performed by utility commands, such as **startstone** to start the stone. These command-line tools accept optional arguments, including such things as process names, port numbers, and log file locations.

Utility commands are listed in Appendix B.

Run-time configurations

Some configuration details are fixed at startup, so making changes requires stopping and restarting the Stone (which stops all other processes except the NetLDI) or Gem (which means logging out). However, other configuration settings can be modified at runtime by executing Smalltalk code.

Example Configurations

As an example of how the scale of application characteristics affect key GemStone configuration settings, the following table provides some examples of “average” configurations.

While these may be useful in planning, the actual values will be based on your particular hardware and application requirements, and most systems will require additional tuning for optimal performance. GemStone can generate performance statistics, both during development and in production systems. These statistics can be displayed and analyzed using the VSD (Visual Statistics Display) utility to understand bottlenecks and determine both changes that are needed, and to study the effect of configuration and application changes.

Table 1.1 Example Configurations

Characteristic or Configuration Option	Sample Server Configuration		
	Small	Medium	Large
Application Characteristics			
Maximum number of user sessions	10	250	1000
Repository size	100 MB	10 GB	500 GB
System Requirements			
Typical number of CPUs	2	4	8+
RAM	4 GB	8 GB	128 GB
Kernel shared memory	2 GB	6 GB	100 GB
Number of disk drives	4	8	12
Configuration Settings			
STN_MAX_SESSIONS	20	280	1200
SHR_PAGE_CACHE_SIZE_KB	75 MB	4 GB	96 GB
Extents			
DBF_EXTENT_NAMES	(1 file)	(4 files)	(10 files)
DBF_EXTENT_SIZES	(unlimited)	(unlimited)	(unlimited)
DBF_ALLOCATION_MODE	SEQUENTIAL	10,10,10,10	10,10,10,...
Transaction Logs			
STN_TRAN_LOG_DIRECTORIES	(1 directory)	(4 directories)	(8 raw partitions)
STN_TRAN_LOG_SIZES	100	500 each	2000 each

GemStone Professional Services can provide expert assistance in establishing your configuration, tuning configurations for performance, and ensuring your configuration can accommodate predicted growth.

More information about the individual settings is provided in the detailed instructions for establishing your own configuration, in Chapter 2, "Configuring the GemStone Server". For details on the specific configuration parameters, see Appendix A.

1.7 Avoiding risk of Data loss

Protecting data against any risk of loss is a critical part of configuration and application design. Power loss, hardware failure, and disk failures are a risk for any critical system and there are a number of ways to ensure data is protected.

Every unexpected shutdown is different, and the details of how to proceed will depend on the specifics; troubleshooting and recovery is described in more detail in Chapter 4.

GemStone's elements that protect against data loss include:

- ▶ Transaction logs allow automatic recovery after a minor failure, such as a power loss.
- ▶ Backups, either programmatic or extent copies, allow you to restore to the time of the backup. Transaction logs in full logging mode allow you to restore transactions.
- ▶ Disk level mirroring, or RAID storage, for the transaction logs protects against failure of the disks holding transaction logs.
- ▶ hot standbys and warm standbys have secondary systems running in parallel, so they can be made available quickly in case of failure in the primary system.

Recovery vs. Restore

Provided whatever caused the error has been corrected and there is no file damage, a Stone startup following an unexpected shutdown will automatically read the transaction log entries that follow the last checkpoint recorded in the repository, and replay these transactions, recovering the GemStone repository with all committed transactions at the point of shutdown. This is termed "recovery".

If the extent disk files are damaged, you may need to restore from backup. This involves using a previously-made backup, and replaying all transaction logs that were generated since the backup was made. For this to succeed, you must have a complete set of backup files, and each of the transaction logs that were generated since the time that the backup was started.

Developing a Failover Strategy

The particular strategy to use to protect your data, depends on what the tolerance for data loss is, how soon your application must be available again following an unexpected outage, and the resources available to support a strategy.

Make regular backups and using full transaction logging mode provides the minimal amount of reliability.

Assuming that backups are kept on a separate disk, mirroring the transaction logs (using OS-level tools) avoids the risk of a disk failure causing loss of committed transactions.

A warm or hot standby system can allow the fastest failover, by keeping a secondary system running at all times. In this case, only a minimal amount of further restore is needed before the standby is usable.

For more on standbys, see Chapter 12, "Warm and Hot Standbys", starting on page 239

Verifying strategy

Backups are a form of insurance; they are essential, but may never actually be needed.

While it is not part of everyday operations, it is imperative that you periodically verify that your backup files are correct and usable, and that you know how to perform the recovery or failover operations. When and if a disk failure or other problem occurs, that will be too late to discover you have no recent backups or the backup process has been failing and the backups you have are unusable.

The pressure of a production-down application is not an ideal time for creating your failover or recovery process, nor for the first time executing any failover processes.

1.8 Running a Second Repository

You can run more than one repository on a single node — for example, separate production and development repositories. There are several points to keep in mind:

- ▶ Each repository requires its own Stone repository monitor process, extent files, transaction logs, and configuration file. Each Stone will start its own shared page cache monitor and a set of other processes, as described on page 24.
- ▶ Multiple Stones that are running the same version of GemStone can share a single installation directory, provided that you create separate repository extents, transaction logs, and configuration files.
- ▶ You must give each Stone a unique name at startup. That name is used to identify the server and to maintain separate log files. Users will connect to the repository by specifying the Stone's name.
- ▶ A single NetLDI typically serves all Stones and Gem session processes on a given node, provided both Stones are running the same version. If you are running two different versions of GemStone, you will need two NetLDIs.

Configuring the GemStone Server

This chapter tells you how to configure the GemStone server processes, repository, transaction logs, and shared page cache to meet the needs of a specific application. It includes the following topics:

Server Components (page 39)

The basics of GemStone server configuration, and setting up basic configuration.

Configuring Extents and Transaction Logs (page 43)

Determining the important parameters for your needs, and how to setup your system.

How To Access the Server Configuration at Run Time (page 55)

Adjustments that can be made to your system while GemStone is running.

Tuning Server Performance (page 58)

Tuning the GemStone server for performance.

This chapter describes configuring the GemStone server; for information about configuring session processes for clients, refer to Chapter 3.

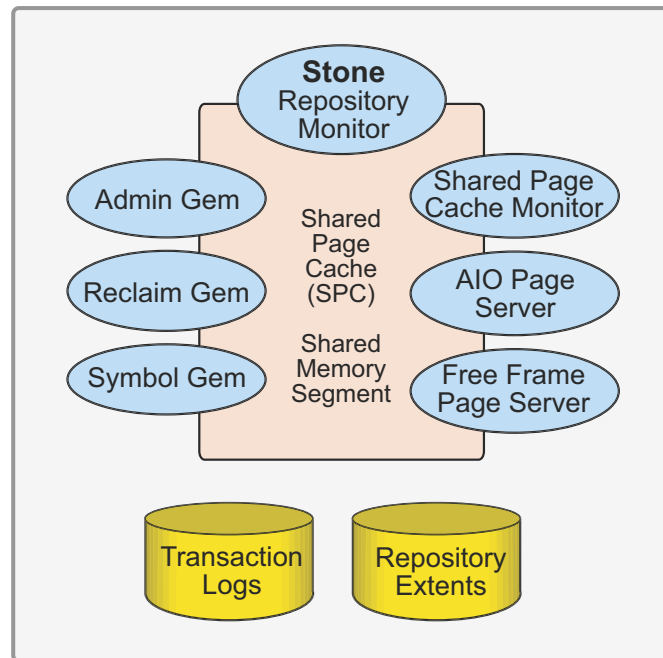
For instructions on starting up the GemStone server, and troubleshooting, see Chapter 6.

2.1 Server Components

The Server is the heart of the GemStone Object Repository. The GemStone server holds the shared classes and objects that compose your data and the shared portion of your application. These classes and objects are written to disk for reliability, and cached in memory for performance. A number of processes work together to keep the GemStone repository running smoothly and support multiple users with different demands and requirements.

The server components are shown in Figure 2.1.

Figure 2.1 The GemStone Server Components



The Server Configuration File

At start-up time, GemStone reads a system-wide configuration file. By default this file is `$GEMSTONE/data/system.conf`, where `GEMSTONE` is an environment variable that points to the directory in which the GemStone software is installed.

Appendix A, “GemStone Configuration Options”, describes the various ways to specify the configuration file to use, and syntax of this file, as well as describing each individual configuration parameter.

There are a large number of configuration options; some are specific to the Stone, some for Gems, others used by all processes. Most parameters have default values that apply if the parameter is not explicitly set, or if there is a syntax error.

Before you begin

To configure GemStone, is necessary to know or estimate the following:

- ▶ The number of simultaneous sessions that will be logged in to the repository. This may include more than one session per user. You should size your system to accommodate the maximum number of users, plus a few extra for administrative logins.
- ▶ The approximate size of your repository, and how much you expect it to grow. For a new system, it may be impossible to come up with a realistic number. The example configurations described in “Example Configurations” on page 35 may give you some idea, but application requirements vary widely.

Number of User Sessions

The maximum number of users is set by the configuration parameter `STN_MAX_SESSIONS`. This defaults to 40, but on startup, the Stone checks for keyfile limits, and if your licensed limit is less than 40, the maximum number of sessions allowed by the keyfile is applied.

Since the Shared Page Cache (SPC) allows space for the maximum number of sessions, setting this much larger than you require creates some unnecessary overhead. However, if at some point you do need more sessions than you have configured as the limit, the cache and stone must be restarted with the larger configuration value.

This limit is for user sessions, and does not need to accommodate logins for GemStone server processes (such as the Garbage Collection Gems). However, some operations that affect the system, including multithreaded administrative operations, may require additional sessions from the user session count.

The `SHR_PAGE_CACHE_NUM_PROCS` is normally derived from `STN_MAX_SESSIONS`, but may be explicitly set on some systems. This limit must accommodate the GemStone system processes as well as the number of user sessions.

Shared Page Cache

The Shared Page Cache (SPC), a shared memory segment that all Server processes attach to, is central to the operation and performance of your repository.

Sizing the shared page cache is an important decision. For performance, the shared cache should be sized as large as possible for the given RAM, while allowing some overhead for other memory requirements; and ensuring that the cache does not swap out of memory.

Each Stone has a single shared page cache on its own node, and may have remote page caches on other nodes in distributed configurations (discussed in detail in Chapter 5). The configurations for remote shared pages caches may be different than for the Stone's shared page cache.

Estimating the Size of the Shared Page Cache

The goal in sizing the shared page cache is to make it large enough to hold the application's working set of objects, to avoid the performance overhead of having to read pages from the disk extents. In addition, it is important that the pages that make up the object table, which holds pointers to the data pages, can fit in the shared page cache.

At the same time, there should be sufficient extra memory available over and above what the shared page cache requires, to allow for other memory caches, heap space, etc.

In general, the shared page cache should be configured at no more than 75% of the available memory, though the actual limit depends on the absolute amount of memory, and factors such as Gem temporary object cache size, and other loads on the system.

- ▶ For maximum performance, the shared page cache can be made large enough to hold the entire repository. After cache warming, sessions should always find the objects in the cache, and never have to perform page reads.
- ▶ Configuring the shared page cache as large as possible up to the size of the repository and about 75% of system RAM is recommended for the best performance.
- ▶ A shared page cache size of less than 10% of your repository size may not hold the entire object table, and is likely to see performance degradation.

Once your application is running, you can refine your estimate of the optimal cache size by monitoring the free space, and use this to tune your configuration. See “Monitoring Performance” on page 138; some relevant statistics are `NumberOfFreeFrames`, `FramesFromFreeList`, and `FramesFromFindFree`.

For example, if you have 2GB of RAM, to set the size of the shared page cache to 1.5 GB:

```
SHR_PAGE_CACHE_SIZE_KB = 1500 MB;
```

Verifying OS support for sufficient Shared Memory

To ensure that your OS has enough memory to handle the desired cache size:

- Step 1.** Ensure that your OS is configured to allow shared memory segments of the intended size, as described in the *Installation Guide* for your platform.
- Step 2.** The shared page cache sizing depends in part on the maximum number of sessions that will be logged in simultaneously. Ensure that the `STN_MAX_SESSIONS` configuration option is set correctly for your application requirements.
- Step 3.** Use GemStone’s `shmem` utility to verify that your OS kernel supports the chosen cache size and number of processes. The command line is

```
$(GEMSTONE)/install/shmem existingFile cacheSizeKB numProcs
```

where:

- ▶ `$(GEMSTONE)` is the directory where the GemStone software is installed.
- ▶ `existingFile` is the name of any writable file, which is used to obtain an id (the file is not altered).
- ▶ `cacheSizeKB` is the `SHR_PAGE_CACHE_SIZE_KB` setting.
- ▶ `numProcs` is the value calculated for `SHR_PAGE_CACHE_NUM_PROCS`. It is computed by `STN_MAX_SESSIONS + STN_MAX_GC_RECLAIM_SESSIONS + SHR_NUM_FREE_FRAME_SERVERS + STN_NUM_LOCAL_AIO_SERVERS + 8`. For an existing system, you can find the computer value in the Stone log file.

For instance, for a 1.5 GB shared cache and `numProcs` calculated using all default configuration settings:

```
% touch /tmp/shmem
% $(GEMSTONE)/install/shmem /tmp/shmem 1500000 52
% rm /tmp/shmem
```

If `shmem` is successful in obtaining a shared memory segment of sufficient size, no message is printed. Otherwise, diagnostic output will help you identify the kernel parameter that needs tuning. The total shared memory requested includes cache overhead for cache space and per-session overhead. The actual shared memory segment in this example would be 104865792 bytes (your system might produce slightly different results).

2.2 Configuring Extents and Transaction Logs

The extents and transaction logs hold the disk-based storage of your GemStone objects. You must configure the location and sizes for these, ensuring that disk I/O does not limit performance.

Recommendations About Disk Usage

If you are using a SAN or a disk using some RAID configurations, the follow discussion is not entirely applicable, since the distribution to physical media is handled for you and GemStone's configuration may not affect I/O contention. However, if you see I/O performance issues you may need to review your disk configuration.

A file system on an SSD is will give the fastest performance for transaction logs, and is easier to manage than raw partitions. The SSD should be an NVMe device or PCIe card or be in a storage array with high bandwidth connection (rather than connected by SATA), if very high IO rates are needed. File system extents on SSD are also highly performant.

If you are setting up a large repository and you do not have a SAN or RAID disks, it is recommended that you configure your system with three or more separate physical drives for improved performance. Performance bottlenecks can occur in reading or writing to the extents or in updating the transaction logs.

When developing your configuration, bear in mind the following guidelines:

1. Keep extents and transaction logs separate from a drive with operating system swap space.
2. Keep the extents and transaction logs separate from each other. You can place multiple logs on the same disk, since only one log file is active at a time.
3. Transaction logs can be placed on SSD drives or raw partitions for the best performance in update-intensive systems. Using raw partitions requires two or more partitions to allow switching.
4. Using multiple extents on multiple drives (provided they are actually separate, not a logical partition) can improve performance, particularly if your repository is large relative to your shared page cache size. To benefit from multiple extents on multiple disks, you must use weighted allocation mode. For details about weighted allocation, see "Allocating Data to Multiple Extents" on page 47.

Raw Partitions

Each raw disk partition is like a single large sequential file, with one extent or one transaction log per partition.

Usually, placing extents on file systems is as fast as using raw partitions. It is possible for this to yield better performance, if it reduces swapping; but your configuration should be designed to avoid swapping in any case.

Placing transaction logs on raw partitions is likely to yield better performance, particularly in an update-intensive application, since such applications primarily are writing sequentially to the active transaction log. Using raw partitions can improve the maximum achievable commit rate by avoiding the extra file system operations necessary to ensure that each log entry is recorded on the disk.

Transaction logs use sequential access exclusively, so the devices can be optimized for that access.

Because each partition holds a single log or extent, if you place transaction logs in raw partitions, you must provide at least two such partitions so that GemStone can preserve one log when switching to the next. The transaction log in each partition must be archived and cleared while the other transaction log is being updated. If your application has a high transaction volume, you are likely to need additional raw partitions for more logs.

For information about using raw partitions, see “How To Set Up a Raw Partition” on page 52.

NFS

Although the Network File System (NFS) can be used to share executables, libraries, and configuration files, they are not recommended, and by default disallowed, for sharing repository extents and tranlogs.

Configuring the Repository Extents

Configuring the repository extents involves these primary considerations:

- ▶ Providing sufficient disk space
- ▶ Minimizing I/O contention
- ▶ Providing fault tolerance

Estimating Extent Size

When you estimate the size of the repository, allow 10 to 20% for fragmentation. Also allow at least 0.5 MB of free space for each session that will be logged in simultaneously. In addition, while the application is running, overhead is needed for objects that are created or that have been dereferenced but are not yet removed from the extents. The amount of extent space required for this depends strongly on the particular application and how it is used.

Reclaim operations and sessions with long transactions may also require a potentially much larger amount of extent free space for temporary data. To avoid the risk of out of space conditions, it is recommended to allow a generous amount of free space.

If there is room on the physical disks, and the extents are not at their maximum sizes as specified using `DBF_EXTENT_SIZES`, then the extents will grow automatically when additional space is needed.

The extent sizes and limits that the system uses are always in multiples of 16MB; using a number that is not a multiple of 16MB results in the next smallest multiple of 16MB being actually used.

If the free space in extents falls below a level set by the `STN_FREE_SPACE_THRESHOLD` configuration option, the Stone takes a number of steps to avoid shutting down. However, if these steps are not effective, the Stone will shut down. For information, see “Recovering from Disk-Full Conditions” on page 196.

For planning purposes, you should allow additional disk space for making GemStone backups and for making a copy of the repository when upgrading to a new release. A GemStone full backup may occupy 75% to 90% of the total size of the extents, depending on how much space is free in the repository at the time.

Choosing the Extent Location

You should consider the following factors when deciding where to place the extents:

- ▶ Keep extents on a spindle different from operating system swap space.
- ▶ Where possible, keep the extents and transaction logs on separate spindles.

Specify the location of each extent in the configuration file. Two examples:

```
DBF_EXTENT_NAMES = $GEMSTONE/data/extent0.dbf
DBF_EXTENT_NAMES = /disk1/gemstone/dbf1.dbf,
                  /disk2/gemstone/dbf2.dbf,
                  /disk3/gemstone/dbf3.dbf;
```

Extent disk configuration

Extents benefit from efficiency of both random access (16 KB repository pages) and sequential access. Don't optimize one by compromising the other. Sequential access is important for such operations as garbage collection and making or restoring backups. Using RAID devices or striped file systems that cannot efficiently support both random and sequential access may reduce overall performance. Simple disk mirroring may be give better results.

Setting a Maximum Size for an Extent

You can specify a maximum size in MB for each extent through the `DBF_EXTENT_SIZES` configuration option. When the extent reaches that size, GemStone stops allocating space in it. If no size is specified, which is the default, GemStone continues to allocate space for the extent until the file system or raw partition is full.

For best performance using raw partitions, the maximum size should be 16MB smaller than the size of the partition, so that GemStone can avoid having to handle system errors. For example, for a 2 GB partition, set the size to 1984 MB.

Each size entry is for the corresponding entry in the `DBF_EXTENT_NAMES` configuration option. Use a comma to mark the position of an extent for which you do not want to specify a limit.

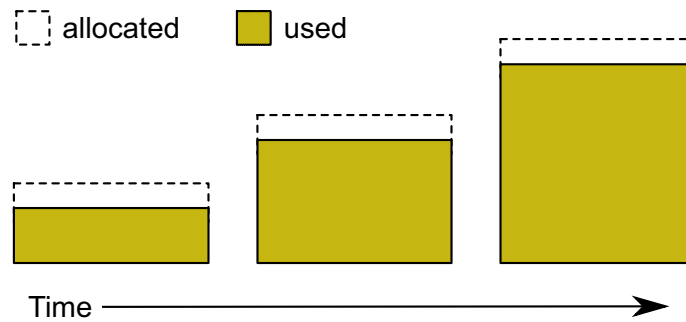
The first example here specifies the size of a single extent, while the second specifies sizes of the first and third extents:

```
DBF_EXTENT_SIZES = 500MB;
DBF_EXTENT_SIZES = 1GB, , 500MB;
```

Pregrowing Extents to a Fixed Size

Allocating disk space requires a system call that introduces run time overhead. Each time an extent is expanded (Figure 2.2), your application must incur this overhead and then initialize the added extent pages.

Figure 2.2 Growing an Extent on Demand



You can increase I/O efficiency while reducing file system fragmentation by instructing GemStone to allocate an extent to a predetermined size (called pregrowing it) at startup.

You can specify a pregrow size for each extent through the `DBF_PRE_GROW` configuration option. When this is set, the Stone repository monitor allocates the specified amount of disk space when it starts up with an extent that is smaller than the specified size. The extent files can then grow as needed up to the limit of `DBF_EXTENT_SIZES`, if that is set, or to the limits of disk space.

Pregrowing extents avoids repeated system calls to allocate and initialize additional space incrementally. This technique can be used with any number of extents, and with either raw disk partitions or extents in a file system.

The disadvantages of pregrowing extents are that it takes longer to start up GemStone, and unused disk space allocated to pregrown extents is unavailable for other purposes.

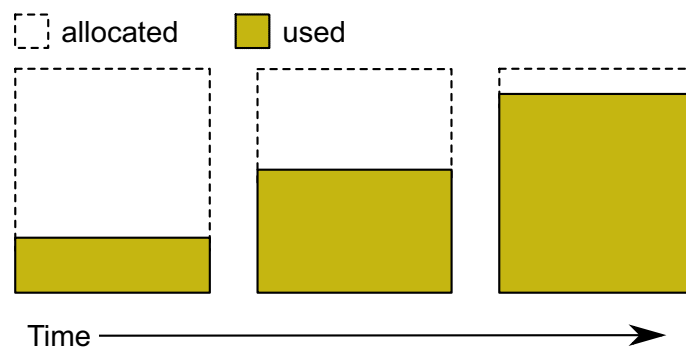
During startup, the Stone starts additional page server processes to handle the pregrow.

Pregrowing Extents to the Maximum Size

You may pregrow extents to the maximum sizes specified in `DBF_EXTENT_SIZES` by setting `DBF_PRE_GROW` to `True`, rather than to a list of pregrow sizes.

Pregrowing extents to the maximum size provides a simple way to reserve space on a disk for a GemStone extent. Since extents cannot be expanded beyond the maximum specified size, the system should be configured with sufficiently large extent sizes that the limit will not be reached, to avoid running out of space.

Figure 2.3 Pregrowing an Extent



Two configuration options work together to pregrow extents. `DBF_PRE_GROW` (page 295) enables the operation, and optionally sets a minimum value to which to size that extent. When `DBF_PRE_GROW` is set to `True`, the Stone repository monitor allocates the space specified by `DBF_EXTENT_SIZES` (page 295) for each extent, when it creates a new extent or starts with an extent that is smaller than the specified size. It may also be set to a list of sizes, which sets the pregrow size individually for each extent to a value that is smaller than `DBF_EXTENT_SIZES`.

For example, to pregrow extents to the maximum size of 1 GB each:

```
DBF_EXTENT_SIZES = 1GB, 1GB, 1GB;  
DBF_PRE_GROW = TRUE;
```

To pregrow extents to 500M, but allow them to later expand to 1 GB if GemStone requires additional space, and that disk space is available:

```
DBF_EXTENT_SIZES = 1GB, 1GB, 1GB;  
DBF_PRE_GROW = 500MB, 500MB, 500MB;
```

Allocating Data to Multiple Extents

Larger applications may improve performance by dividing the repository into multiple extents. Assuming the extents are on multiple spindles or the disk controller manages files as if they were, this allows several extents to be active at once.

The setting for the `DBF_ALLOCATION_MODE` configuration option determines whether GemStone allocates new disk pages to multiple extents by filling each extent sequentially or by balancing the extents using a set of weights you specify. Weighted allocation yields better performance because it distributes disk accesses.

Sequential Allocation

By default, the Stone repository monitor allocates disk resources sequentially by filling one extent to capacity before opening the next extent. (See Figure 2.4 on page 48) For example, if a logical repository consists of three extents named A, B, and C, then all of the disk resources in A will be allocated before any disk resources from B are used, and so forth. Sequential allocation is used when the `DBF_ALLOCATION_MODE` configuration option is set to `SEQUENTIAL`.

Weighted Allocation

For weighted allocation, you use `DBF_ALLOCATION_MODE` to specify the number of extent pages to be allocated from each extent on each allocation request. The allocations are positive integers in the range 1..40 (inclusive), with each element corresponding to an extent of `DBF_EXTENT_NAMES`. For example:

```
DBF_EXTENT_NAMES = a.dbf, b.dbf, c.dbf;  
DBF_ALLOCATION_MODE = 12, 20, 8;
```

You can think of the total weight of a repository as the sum of the weights of its extents. When the Stone allocates space from the repository, each extent contributes an allocation proportional to its weight.

NOTE

We suggest that you avoid using very small values for weights, such as "1,1,1". It's more efficient to allocate a group of pages at once, such as "10,10,10", than to allocate single pages repeatedly.

One reason for specifying weighted allocation is to share the I/O load among a repository's extents. For example, you can create three extents with equal weights, as shown in Figure 2.5 on page 48.

Figure 2.4 Sequential Allocation

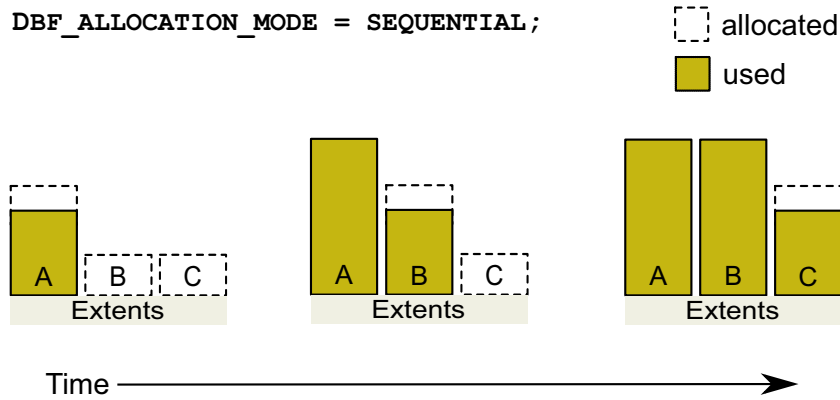
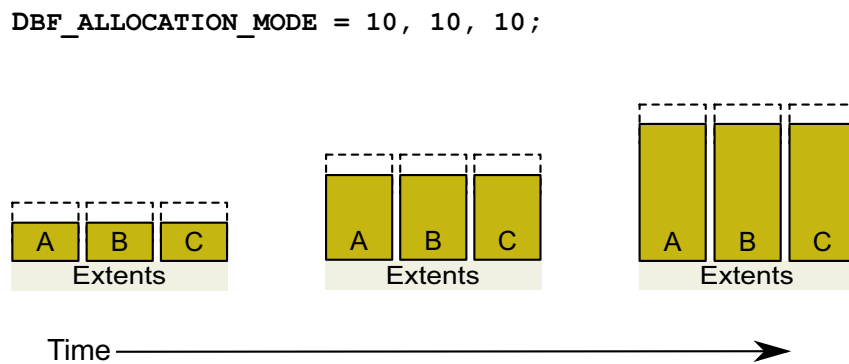


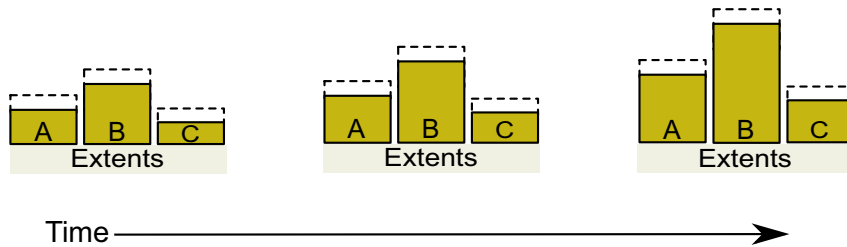
Figure 2.5 Equally Weighted Allocation



Although equal weights are most common, you can adjust the relative extent weights for other reasons, such as to favor a faster disk drive. For example, suppose we have defined three extents: A, B, and C. If we defined their weights to be 12, 20, and 8 respectively, then for every 40 disk units (pages) allocated, 12 would come from A, 20 from B, and 8 from C. Another way of stating this formula is that because B's weight is 50% of the total repository weight, 50% of all newly-allocated pages are taken from extent B. Figure 2.6 shows the result.

Figure 2.6 Proportionally Weighted Allocation

```
DBF_ALLOCATION_MODE = 12, 20, 8;
```



You can modify the relative extent weights by editing your GemStone configuration file to modify the values for `DBF_ALLOCATION_MODE`, or to change `DBF_ALLOCATION_MODE` to `SEQUENTIAL`; the new allocation scheme take effect the next time you start GemStone. The changes do not affect existing page allocations, only how pages are allocated for ongoing page requirements.

Effect of Clustering on Allocation Mode

Explicit clustering of objects using instances of `ClusterBucket` that explicitly specify an `extentId` takes precedence over `DBF_ALLOCATION_MODE`. For information about clustering objects, refer to the *Programming Guide*.

Weighted Allocation for Extents Created at Run Time

Smalltalk methods for creating extents at run time (`Repository>>createExtent:` and `Repository>>createExtent:withMaxSize:`) do not provide a way to specify a weight for the newly-created extent. If your repository uses weighted allocation, the Stone repository monitor assigns the new extent a weight that is the simple average of the repository's existing extents. For instance, if the repository is composed of three extents with weights 6, 10, and 20, the default weight of a newly-created fourth extent would be 12 (36 divided by 3).

Configuring the Transaction Logs

Configuring the transaction logs involves considerations similar to those for extents:

- ▶ Providing sufficient disk space
- ▶ Minimizing I/O contention
- ▶ Providing fault tolerance, through the choice of logging mode

Logging Mode

GemStone provides two modes of transaction logging:

- ▶ *Full logging*, the default mode, provides real-time incremental backup of the repository. Deployed applications should use this mode. All transactions are logged regardless of their size, and the resulting logs can used in restoring the repository from a GemStone backup.

- ▶ *Partial logging* is intended for use during evaluation or early stages of application development. Partial logging allows a simple operation to run unattended for an extended period and permits automatic recovery from system crashes that do not corrupt the repository. Logs created in this mode cannot be used in restoring the repository from a backup.

In partial logging mode, frequent backups are recommended; any data that is not backed up may be lost if there are disk issues or the repository becomes corrupted.

Full logging is “sticky”; once a repository has started in full logging, changing the logging back to partial logging requires special steps. See “To Change to Partial Logging” on page 207. If you are in partial logging mode, to enable full transaction logging, simply change the configuration setting `STN_TRAN_FULL_LOGGING` to True and restart the Stone.

After changing from partial to full logging, make a backup as soon as possible. While you may have backups that were made in partial logging mode, these partial-logging-mode backups cannot be used to restore the full-logging-mode transaction logs.

For more information about the logging mode and the administrative differences, see “Logging Modes” on page 201.

Estimating Disk Space for Transaction Logs

How much disk space does your application need for transaction logs? The answer depends on several factors:

- ▶ The logging mode that you choose
- ▶ Characteristics of your transactions
- ▶ How often you archive and remove the logs

If GemStone is in full transaction logging mode (the default), you must allow sufficient space to log all transactions until you next archive the logs.

CAUTION

If the Stone exhausts the transaction log space, users will be unable to commit transactions until space is made available.

You can estimate the space required from your transaction rate and the number of bytes modified in a typical transaction. Example 2.1 provides an estimate for an application that expects to generate 4500 transactions a day.

GemStone is writing to exactly one log at any given time. The number of transaction logs that should be available for automatic recovery will include this log, of course, and zero or more previous logs. The number of previous logs that is needed depends on the time of the most recent checkpoint, and if there are any long-running transactions.

The method `Repository>>oldestLogFileIdForRecovery` identifies the oldest log file needed for recovery from the most recent checkpoint, if the Stone were to crash. These log files should be left in place to allow automatic recovery.

Log files older than this are needed only if it becomes necessary to restore the repository from a backup. These may be archived, but should be kept at least until the next backup is made.

Example 2.1 Space for Transaction Logs Under Full Logging

Average transaction rate	= 5 per minute
Duration of transaction processing	= 15 hours per day
Average transaction size	= 5 KB
Archiving interval	= Daily
Transactions between archives	
5 per minute * 60 minutes * 15 hours	= 4500
Log space (minimum)	
4500 transactions * 5 KB	= 22 MB

If GemStone is configured for partial logging, you need only provide enough space to maintain transaction logs since the last repository checkpoint. Ordinarily, two log files are sufficient: the current log and the immediately previous log. There is no need to retain logs; in partial logging mode, transaction logs are used only after an unexpected shutdown to recover transactions since the last checkpoint.

Choosing the Log Location and Size Limit

The considerations in choosing a location for transaction logs are similar to those for extents:

- ▶ Keep transaction logs on a different disk than operating system swap space.
- ▶ Where possible, keep the extents and transaction logs on separate disks – doing so reduces I/O contention while increasing fault tolerance.
- ▶ Because update-intensive applications primarily are writing to the transaction log, storing raw data in a disk partition (rather than in a file system) may yield better performance.

WARNING

Because the transaction logs are needed to recover from a system crash, do NOT place them in directories such as /tmp that are automatically cleared during power-up.

Transaction logs use sequential access exclusively, so the devices can be optimized for that access.

With raw partitions, or when in partial transaction logging mode, GemStone requires at least two log locations (directories or raw partitions) so it can switch to another when the current one is filled. In full transaction mode, logging to transaction logs on the file system, one directory may be used, in which case all transaction logs are created in that directory.

When you set the log locations in the configuration file, you should also check their size limit.

Although the size of 100 MB provided in the default configuration file is adequate in many situations, update-intensive applications should consider a larger size to limit the

frequency with which logs are switched. Each switch causes a checkpoint to occur, which can impact performance.

NOTE

For best performance using raw partitions, the size setting should be slightly smaller than the size of the partition so GemStone can avoid having to handle system errors. For example, for a 2 GB partition, set it to 1998 MB.

The following example sets up a log in a 2 GB raw partition and a directory of 100 MB logs in the file system. This setup is a workable compromise when the number of raw partitions is limited. The file system logs give the administrator time to archive the primary log when it is full.

```
STN_TRAN_LOG_DIRECTORIES = /dev/rdisk/c4d0s2, /user3/tranlogs;  
STN_TRAN_LOG_SIZES = 1998, 100;
```

All of the transaction logs must reside on Stone's node.

How To Set Up a Raw Partition

WARNING

Using raw partitions requires extreme care. Overwriting the wrong partition destroys existing information, which in certain cases can make data on the entire disk inaccessible.

Raw partitions are system-dependent; you will need to work with your system administrator and consult your system documentation to setup or locate a partition of suitable size.

You can mix file system-based files and raw partitions in the same repository, and you can add a raw partition to existing extents or transaction log locations. The partition reference in `/dev` must be readable and writable by anyone using the repository, so you should give the entry in `/dev` the same protection as you would use for the corresponding type of file in the file system.

The first step is to find a partition (raw device) that is available for use. Depending on your operating system, a raw partition may have a name like `/dev/rdisk/c1t3d0s5`, `/dev/rsd2e`, or `/dev/vg03/r1v0l1`. A partition is available if all of the following are true:

- ▶ It does not contain the root (`/`) file system (on some systems, the root volume group).
- ▶ It is not on a device that contains swap space.
- ▶ Either it does not contain a file system or that file system can be left unmounted and its contents can be overwritten.
- ▶ It is not already being used for raw data.

When you select a partition, make sure that any file system tables, such as `/etc/vfstab`, do not call for it to be mounted at system boot. Use `chmod` and `chown` to set read-write permissions and ownership of the special device file the same way you would protect a repository file in a file system. .

If the partition will contain the primary extent (the first or only one listed in `DBF_EXTENT_NAMES`), initialize it by using the GemStone `copydbf` utility to copy an existing repository extent to the device. The extent must not be in use when you copy it. If

the partition already contains a GemStone file, first use **removedbf** to mark the partition as being empty.

Partitions for transaction logs do not need to be initialized, nor do secondary extents into which the repository will expand later.

Sample Setup for Extent on Raw Partition

The following example configures GemStone to use the raw partition `/dev/rsd2d` as the repository extent.

Step 1. If the raw partition already contains a GemStone file, mark it as being empty. (The **copydbf** utility will not overwrite an existing repository file.)

```
% removedbf /dev/rsd2d
```

Step 2. Use **copydbf** to install a fresh extent on the raw partition. (If you copy an existing repository, first stop any Stone that is running on it, or suspend checkpoints)

```
% copydbf $GEMSTONE/bin/extent0.dbf /dev/rsd2d
```

Step 3. As root, change the ownership and the permission of the partition special device file in `/dev` to what you ordinarily use for extents in a file system. For instance:

```
# chown gsAdmin /dev/rsd2d
# chmod 600 /dev/rsd2d
```

You should also consider restricting the execute permission for `$GEMSTONE/bin/removedbf` and `$GEMSTONE/bin/removeextent` to further protect your repository. In particular, these executable files should not have the setuid (S) bit set.

Step 4. Edit the Stone's configuration file to show where the extent is located:

```
DBF_EXTENT_NAMES = /dev/rsd2d;
```

Step 5. Use **startstone** to start the Stone repository monitor in the usual manner.

Changing Between Files and Raw Partitions

This section tells you how to change your configuration by moving existing repository extent files to raw partitions or by moving existing extents in raw partitions to files in a file system. You can make similar changes for transaction logs.

Moving an Extent to a Raw Partition

To move an extent from the file system to a raw partition, do this:

Step 1. Define the raw disk partition device. Its size should be at least 16 MB larger than the existing extent file.

Step 2. Stop the Stone repository monitor.

Step 3. Edit the repository's configuration file, substituting the device name of the partition for the file name in `DBF_EXTENT_NAMES` (page 294).

Set `DBF_EXTENT_SIZES` for this extent to be 16 MB smaller than the size of the partition.

Step 4. Use **copydbf** to copy the extent file to the raw partition. (If the partition previously contained a GemStone file, first use **removedbf** to mark it as unused.)

Step 5. Restart the Stone.

Moving an Extent to the File System

The procedure to move an extent from a raw partition to the file system is similar:

Step 1. Stop the Stone repository monitor.

Step 2. Edit the repository's configuration file, substituting the file pathname for the name of the partition in `DBF_EXTENT_NAMES`.

Step 3. Use `copydbf` to copy the extent to a file in a file system, then set the file permissions to the ones you ordinarily use.

Step 4. Restart the Stone.

Moving Transaction Logging to a Raw Partition

To switch from transaction logging in the file system to logging in a raw partition, do this:

Step 1. Define the raw disk partition. If you plan to copy the current transaction log to the partition, its size should be at least 1 to 2 MB larger than current log file.

Step 2. Stop the Stone repository monitor.

Step 3. Edit the repository's configuration file, substituting the device name of the partition for the directory name in `STN_TRAN_LOG_DIRECTORIES` (page 330). Make sure that `STN_TRAN_LOG_SIZES` for this location is 1 to 2 MB smaller than the size of the partition.

Step 4. Use `copydbf` to copy the current transaction log file to the raw partition. (If the partition previously contained a GemStone file, first use `removedbf` to mark it as unused.)

You can determine the current log from the last message "Creating a new transaction log" in the Stone's log. If you don't copy the current transaction log, the Stone will open a new one with the next sequential fileId, but it may be opened in another location specified by `STN_TRAN_LOG_DIRECTORIES`.

Step 5. Restart the Stone.

Moving Transaction Logging to the File System

The procedure to move transaction logging from a raw partition to the file system is similar:

Step 1. Stop the Stone repository monitor.

Step 2. Edit the repository's configuration file, substituting a directory pathname for the name of the partition in `STN_TRAN_LOG_DIRECTORIES`.

Step 3. Use `copydbf` to copy the current transaction log to a file in the specified directory. The `copydbf` utility will generate a file name like `tranlognnn.dbf`, where `nnn` is the internal fileId of that log.

Step 4. Restart the Stone.

Server Response to Gem Fatal Errors

The Stone repository monitor is configurable in its response to a fatal error detected by a Gem session process. If configured to do so, the Stone can halt and dump debug information if it receives notification from a Gem that the Gem process died with a fatal error. By stopping both the Gem and the Stone at this point, the possibility of repository corruption is minimized.

In the default mode, the Stone does not halt if a Gem encounters a fatal error. This is usually preferable for deployed production systems.

During application development, it may be helpful to know exactly what the Stone was doing when the Gem went down. It may in some cases be preferred to absolutely minimize the risk of repository corruption, at the risk of system outage. To configure the Stone to halt when a fatal gem error is encountered, change the following in the Stone's configuration file:

```
STN_HALT_ON_FATAL_ERR = TRUE;
```

2.3 How To Access the Server Configuration at Run Time

GemStone provides several methods in class System that let you examine, and in certain cases modify, the configuration parameters at run time from Smalltalk.

To Access Current Settings at Run Time

Class methods in System, in the in category Configuration File Access, let you examine the system's Stone configuration. The following access methods all provide similar server information:

`stoneConfigurationReport`

Returns a SymbolDictionary whose keys are the names of configuration file parameters, and whose values are the current settings of those parameters in the repository monitor process.

`configurationAt: aName`

Returns the value of the specified configuration parameter, giving preference to the current session process if the parameter applies to a Gem.

`stoneConfigurationAt: aName`

Returns the value of the specified configuration parameter from the Stone process, or returns `nil` if that parameter is not applicable to a Stone.

(The corresponding methods for accessing a session configuration are described on page 68.)

Here is a partial example of the Stone configuration report:

```
topaz 1> printit
System stoneConfigurationReport asReportString
%
#'StnEpochGcEnabled'    false
#'StnCrBacklogThreshold'      80
#'STN_TRAN_LOG_SIZES'    100
#'StnTranLogDebugLevel'  0
...

```

Keys in mixed capitals and lowercase, such as `StnEpochGcEnabled`, are internal run-time parameters.

To Change Settings at Run Time

The class method `System class>>configurationAt: aName put: aValue` lets you change the value of the internal run-time parameters in Table 2.1, if you have the appropriate privileges.

In the reports described in the preceding section, parameters with names in all uppercase are read-only; for parameters that can be changed at runtime, the name is in mixed case.

CAUTION

Avoid changing configuration parameters unless there is a clear reason for doing so. Incorrect settings can have serious adverse effects on performance. For additional guidance about run-time changes to specific parameters, see Appendix A, “GemStone Configuration Options”.

Table 2.1 Server Configuration Parameters Changeable at Run Time

Configuration File Option	Internal Parameter	Required Privilege
SHR_SPIN_LOCK_COUNT	#SpinLockCount	Login as SystemUser
STN_ADMIN_GC_SESSION_ENABLED	#StnAdminGcSessionEnabled	GarbageCollection
STN_CHECKPOINT_INTERVAL	#StnCheckpointInterval	Login as SystemUser
STN_COMMIT_QUEUE_THRESHOLD	#StnCommitQueueThreshold	Login as SystemUser
STN_CR_BACKLOG_THRESHOLD	#StnCrBacklogThreshold	Login as SystemUser
STN_DISABLE_LOGIN_FAILURE_LIMIT	#StnDisableLoginFailureLimit	OtherPassword
STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT	#StnDisableLoginFailureTimeLimit	OtherPassword
STN_DISKFULL_TERMINATION_INTERVAL	#StnDiskFullTerminationInterval	Login as SystemUser
STN_EPOCH_GC_ENABLED	#StnEpochGcEnabled	GarbageCollection
STN_FREE_SPACE_THRESHOLD	#StnFreeSpaceThreshold	Login as SystemUser
STN_GEM_ABORT_TIMEOUT	#StnGemAbortTimeout	Login as SystemUser
STN_GEM_LOSTOT_TIMEOUT	#StnGemLostOfTimeout	Login as SystemUser
STN_GEM_TIMEOUT	#StnGemTimeout	Login as SystemUser
STN_HALT_ON_FATAL_ERR	#StnHaltOnFatalErr	Login as SystemUser
STN_LOG_LOGIN_FAILURE_LIMIT	#StnLogLoginFailureLimit	OtherPassword

Table 2.1 Server Configuration Parameters Changeable at Run Time (Continued)

Configuration File Option	Internal Parameter	Required Privilege
STN_LOG_LOGIN_FAILURE_TIME_LIMIT	#StnLogLoginFailureTimeLimit	OtherPassword
STN_LOOP_NO_WORK_THRESHOLD	#StnLoopNoWorkThreshold	Login as SystemUser
STN_MAX_AIO_RATE	#StnMntMaxAioRate	Login as SystemUser
STN_MAX_LOGIN_LOCK_SPIN_COUNT	#StnMaxLoginLockSpinCount	SystemControl
STN_MAX_VOTING_SESSIONS	#StnMaxVotingSessions	Login as SystemUser
STN_NUM_GC_RECLAIM_SESSIONS	#StnNumGcReclaimSessions	GarbageCollection
STN_OBJ_LOCK_TIMEOUT	#StnObjLockTimeout	SystemControl
STN_PAGE_MGR_COMPRESSION_ENABLED	#StnPageMgrCompressionEnabled	Login as SystemUser
STN_PAGE_MGR_MAX_WAIT_TIME	#StnPageMgrMaxWaitTime	Login as SystemUser
STN_PAGE_MGR_PRINT_TIMEOUT_THRESH- OLD	#StnPageMgrPrintTimeoutThresh- old	Login as SystemUser
STN_PAGE_MGR_REMOVE_MAX_PAGES	#StnPageMgrRemoveMaxPages	Login as SystemUser
STN_PAGE_MGR_REMOVE_MIN_PAGES	#StnPageMgrRemoveMinPages	Login as SystemUser
STN_REMOTE_CACHE_PGSRV_TIMEOUT	#StnRemoteCachePgsvrTimeout	SystemControl
STN_REMOTE_CACHE_TIMEOUT	#StnRemoteCacheTimeout	Login as SystemUser
STN_SHR_TARGET_PERCENT_DIRTY	#ShrPcTargetPercentDirty	Login as SystemUser
STN_SIGNAL_ABORT_CR_BACKLOG	#StnSignalAbortCrBacklog	GarbageCollection
STN_SMC_SPIN_LOCK_COUNT	#StnSmcSpinLockCount	SystemControl
STN_SYMBOL_GC_ENABLED	#StnSymbolGcEnabled	GarbageCollection
STN_TRAN_LOG_DEBUG_LEVEL	#StnTranLogDebugLevel	SystemControl
STN_TRAN_LOG_LIMIT	#StnTranLogLimit	Login as SystemUser
STN_TRANQ_TO_RUNQ_THRESHOLD	#StnTranqToRunqThreshold	Login as SystemUser
(none)	#StnLoginsSuspended	SystemControl

The following example first obtains the value of #StnAdminGcSessionEnabled. This value can be changed at run time by a user with GarbageCollection privilege:

```
topaz 1> printit
System configurationAt: #StnAdminGcSessionEnabled
%
true

topaz 1> printit
System configurationAt: #StnAdminGcSessionEnabled put: false
%
false
```

For more information about these methods, see the comments in the image.

2.4 Tuning Server Performance

There are a number of configuration options by which you can tune the GemStone server. These options can help make better use of resources such as memory and I/O.

Tuning the Shared Page Cache

Two configuration options can help you tailor the shared page cache to the needs of your application: `SHR_PAGE_CACHE_SIZE_KB` and `SHR_SPIN_LOCK_COUNT`.

You may also want to consider object clustering within Smalltalk as a means of increasing cache efficiency.

Adjusting the Cache Size

As your repository or number of users grows, you may want to adjust the `SHR_PAGE_CACHE_SIZE_KB` configuration parameter setting.

You can use the statistics for a running application to monitor the load on the cache. In particular, the statistics `FreeFrameCount` and `FramesFromFindFree` may be useful, as well as `FramesFromFreeList`.

As described under “Estimating the Size of the Shared Page Cache” on page 41, increasing the size of the cache, up to the size that will hold the entire repository, will usually improve performance.

Matching Spin Lock Limit to Number of Processors

The setting for the `SHR_SPIN_LOCK_COUNT` configuration option specifies the number of times a process should attempt to obtain a lock in the shared page cache using the spin lock mechanism before resorting to setting a semaphore and sleeping. We recommend leaving `SHR_SPIN_LOCK_COUNT` set to -1 (the default), which causes GemStone to determine whether multiple processors are installed and set the parameter accordingly.

Clustering Objects That Are Accessed Together

Appropriate clustering of objects by the application can allow a smaller shared page cache by reducing the number of data pages in use at once. For general information about clustering objects, see the *Programming Guide*.

Controlling Checkpoint Frequency

On each commit, committed changes are immediately written to the transaction logs, but the writing of this data, recorded on "dirty pages," from the shared page cache to the extents may lag behind.

At a checkpoint, all remaining committed changes that have not yet been written to the extents are written out, and the repository updated to a new consistent committed state. If the volume of these waiting committed changes is high, there may be a performance hit as this data is written out. Between checkpoints, new committed changes written to the extents are not yet considered part of the repository's consistent committed state until the next checkpoint.

If checkpoints interferes with other GemStone activity, you may want to adjust their frequency.

- ▶ In full transaction logging mode, most checkpoints are determined by the `STN_CHECKPOINT_INTERVAL` configuration option, which by default is five minutes. A few Smalltalk methods, such as `Repository>>fullBackupTo:`, force a checkpoint at the time they are invoked. A checkpoint also is performed each time the Stone begins a new transaction log.
- ▶ In partial logging mode, checkpoints also are triggered by any transaction that is larger than `STN_TRAN_LOG_LIMIT` (page 330), which sets the size of the largest entry that is to be appended to the transaction log. The default limit is 1 MB of log space. If checkpoints are too frequent in partial logging mode, it may help to raise this limit. Conversely, during bulk loading of data with large transactions, it may be desirable to lower this limit to avoid creating large log files.

A checkpoint also occurs each time the Stone repository monitor is shut down gracefully, as by invoking `stopstone` or `System class>>shutDown`. This checkpoint permits the Stone to restart without having to recover from transaction logs. It also permits extent files to be copied in a consistent state.

While less frequent checkpoints may improve performance in some cases, they may extend the time required to recover after an unexpected shutdown. In addition, since checkpoints are important in the recycling of repository space, less frequent checkpoints can mean more demand on free space (extent space) in the repository.

Tuning Page Server Behavior

GemStone uses page servers for three purposes:

- ▶ To write dirty pages to disk.
- ▶ To add free frames to the free frame list, from which a Gem can take as needed.
- ▶ To transfer pages from the Stone host to the shared page cache host, if different.

The *AIO page server* is a type of page server that performs all three functions. The AIO page server is running at all times, and is required in order to write updated data (dirty pages) to disk. The default configuration starts only one thread within the AIO page server. Larger applications with multiple extents will want to configure a larger number of threads to avoid a performance bottleneck.

The AIO page server runs only on the Stone's node. By default on the Stone's node, and on all remote nodes, there is a *free frame page server*. The Free frame page server is dedicated only to the third task listed above: adding free frames to the free list.

The number of free frame page server threads defaults to the same number as the number of AIO page server threads, and should not be set to a value lower than that.

Remote cache page servers and *remote Gem page servers* handle the tasks of handling pages between the Stone's node and the remote shared page cache. These page servers will be described in Chapter 5, "Connecting Distributed Systems".

To Add AIO Page Server threads

By default the Stone starts the AIO page server process with a single thread, to perform asynchronous I/O (AIO) between the shared page cache and the extents. This process updates extents during a checkpoint.

If your configuration has multiple extents on separate disk spindles, you should generally increase the number of threads in the AIO page server. You can do this by changing the `STN_NUM_LOCAL_AIO_SERVERS` configuration option.

For multiple page server threads to improve performance, they must be able to execute at the same time and write to disk at the same time. If you have only one CPU, or your extents are on a single disk spindle, multiple AIO page server threads will not be able to write pages out faster than a single thread.

Free Frame Page Server

A Gem can get free frames either from the free list (the quick way), or, if sufficient free frames have not been added to the list, by scanning the shared page cache for a free frame instead. What constitutes sufficient free frames is determined by the `GEM_FREE_FRAME_LIMIT` configuration option).

To assist the AIO page server in adding frames back to the free list, the stone spawns a free frame page server, with the same number of threads as the AIO page server. The number of free frame page server threads should not be less than the number of AIO page server threads, to ensure the distribution of free pages and used pages remains balanced over the repository extents.

In some cases, increasing the number of free frame page servers can improve overall system performance. For example, if Gems are performing many operations requiring writing pages to disk, the AIO page server may have to spend all its time writing pages, never getting a chance to add free frames to the free list. Alternatively, if Gems are performing operations that require only reading, the AIO page server will see no dirty frames in the cache – the signal that prompts it to take action. In that case, it may sleep for a second, even though free frames are present in the cache and need to be added to the free list.

Process Free Frame Caches

There is a communication overhead involved in getting free frames from the free frame list for scanning. To optimize this, you can configure the Gems and their remote page servers to add or remove multiple free frames from a free frame cache to the free frame list in a single operation.

When using the free frame cache, the Gem or remote page server removes enough frames from the free list to refill the cache in a single operation. When adding frames to the free list, the process does not add them until the cache is full.

You can control the size of the Gem and remote page server free frame caches by setting the configuration parameters `GEM_FREE_FRAME_CACHE_SIZE` (page 297) and `GEM_PGSVR_FREE_FRAME_CACHE_SIZE` (page 300), respectively.

The default behavior depends on the size of the shared page cache; if the shared page cache is 100MB or larger, a page server free frame cache size of 10 is used, so ten free frames are acquired in one operation when the cache is empty. For shared page cache sizes less than 100MB, the Gem or remote page server acquires the frames one at a time.

Running Cache Warming

When the repository is first started up, neither the object table nor any of the objects in the extents are loaded into memory. Initial accesses require reading a number of pages from disk containing the object table, and then reading the data pages that contain the objects.

This cost in a freshly restarted repository can be avoided by warming the cache—loading the object table, and optionally data pages containing the objects, into the cache. This allows the cost of loading pages into the cache to be done at startup time, rather than in the course of end-user application queries.

Cache warming is done using the **startcachewarmer** utility. This utility can be run manually, but since it is most useful immediately after startup, you normally configure your system to run this automatically as part of startup. Cache warming may take some minutes to complete, depending on the number of pages and disk performance; you can execute **startstone** and **waitstone** with arguments so they wait for cache warming to complete.

Configure what is loaded into the cache

The decision on what to load into the cache depends on the size of your shared page cache, relative to the size of your repository.

- ▶ If your shared page cache is very large and sufficiently large to hold all objects in the entire repository, you may wish to load all data pages into the cache. This is done using the **-d** argument.
- ▶ If your shared page cache is small relative to the repository size, and if the set of data pages that are used is highly variable, you may wish to load only the object table into the shared cache, and allow data pages to be loaded as needed. Ensure that the working set file `/opt/gemstone/locks/<stoneName><hostid>workingSet.lz4` does not exist, and do not use either the **-d** or **-D** argument.
- ▶ When the entire repository does not fit into the cache, and there is a subset of data that is frequently used, and will predictably be needed after a Stone restart, you can configure the shared page cache monitor to write out a compressed set of pageIds of data pages that are in the cache. The cache warmer looks for this file and automatically loads these data pages. When the **-w** option was specified on cache warming before the previous Stone shutdown, the pages in the shared page cache after restart match what was in the cache on Stone shutdown.

The **-w <interval>** enables writing the working set file to disk, at the specified interval, to the file

```
/opt/gemstone/locks/<stoneName><hostid>workingSet.lz4
```

When the working set file exists, it is automatically used to load pages. Only pages that are valid are loaded, so there is no requirement to keep the file strictly in sync with the repository.

The full set of options for cache warming are described under **startcachewarmer** on page 347.

Configure when warming is run

To perform cache warming automatically every time the repository starts up, use the configuration parameters:

- ▶ `STN_CACHE_WARMER_ARGS` (page 312) to warm the Stone's shared page cache
- ▶ `GEM_CACHE_WARMER_ARGS` (page 296) and `GEM_CACHE_WARMER_MID_CACHE_ARGS` (page 297) to warm shared page caches that are started by Gems.

These parameters take a string containing the specific arguments to the `startcachewarmer` utility. Not all arguments make sense in the context of the configuration parameter; for example, you never need to provide the stone name or configuration file paths.

For example, to configure cache warming to write the working set out once a day and on stone shutdown, and to use four sessions to perform the warming, use the following:

```
STN_CACHE_WARMER_ARGS = "-w 1440 -n 4" ;
```

You may manually execute the `startcachewarmer` utility on the command line at any time, but it is most useful immediately after startup of the page cache to be warmed. Cache warming will only load pages until the cache is full.

Configuring Gem Session Processes

This chapter tells how to configure the GemStone/S 64 Bit session processes for your application. For additional information about running session processes on a node remote from the Stone repository monitor, refer also to Chapter 5.

Overview (page 63)

An overview of Gem sessions.

Configuring Gem Session Processes (page 65)

Configuring the Gem within the GemStone network.

Set the Gem Configuration Options (page 67)

Configuration options for the Gem itself.

How To Access the Configuration at Run Time (page 68)

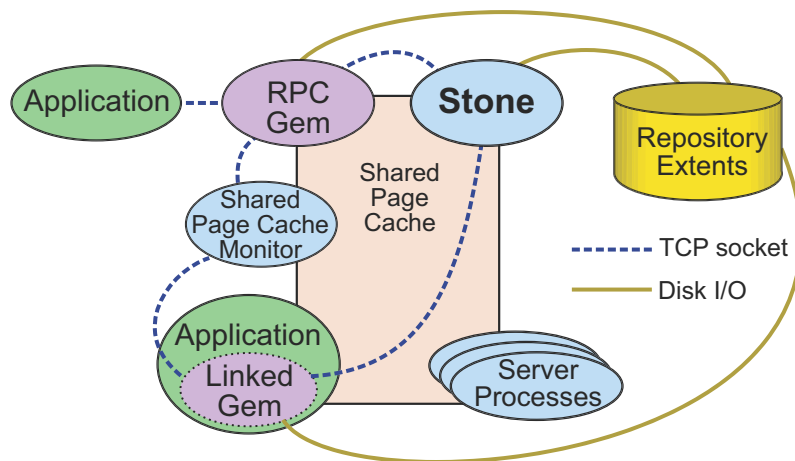
Adjustments that can be made to the Gem settings while the Gem is logged in.

3.1 Overview

As shown in Figure 3.1, a GemStone session involves the following components in a client-server relationship:

- ▶ The user's client Application
- ▶ The Gem, which acts as a server for a particular application. The Gem can be a separate session process, or linked to the client application.
- ▶ The Stone repository monitor, and other server processes.
- ▶ The shared page cache (shared memory segment), and the shared page cache monitor process.
- ▶ The repository extent files

Figure 3.1 GemStone Session Elements



From the point of view of the application, the Gem is the object server; it provides the bulk of the repository capabilities:

- ▶ It logs in to the repository with the Stone, and it obtains object locks, free object identifiers, and free pages from the Stone.
- ▶ It presents the application with a consistent view of the repository during a transaction, and tracks which objects the application accesses.
- ▶ It executes Smalltalk methods within the repository.
- ▶ It reads the repository as the application accesses objects, and (with the help of the AIO page server) it updates the repository when the application successfully commits a transaction.

Linked and RPC Applications

The Gem session process can be run as a separate process (the RPC Gem in Figure 3.1) or integrated with the application into a single process, in which case the application is called a *linked* application (the Linked Gem in Figure 3.1).

When the Gem runs as a separate process, it responds to Remote Procedure Calls (RPCs) from the application; this is called an *RPC* application. Applications that use a separate Gem process start that process automatically (from the user's viewpoint) while logging in to the repository.

Either type of application can be used on a single node or across a network.

An application can have only one linked login, since only one Gem can be integrated with the process. Any application may have multiple RPC logins, or one linked and multiple RPC logins.

GemStone provides both linked and RPC versions of topaz for repository administration, and shared client libraries that allow linked and RPC. The linked version allows both linked and RPC logins, but the RPC version allows only RPC logins.

C programmers should always use an RPC version during development and debugging to protect Gem data structures from possible corruption.

Note on terminology

The term “Gem” is used to refer to both the linked and RPC sessions, although only one of these is a separate OS process. Both normally represent a logged in session in GemStone, and from the Stone’s point of view, there is little difference.

The term “session” is also sometimes used interchangeably with “Gem”. However, multi-threaded operations such as for backup and garbage collection may allow multiple threads within a Gem process to log in individual sessions.

The Session Configuration File

At start-up time, each Gem session process looks for a configuration file, which by default is the same system-wide configuration file used by the repository monitor when it starts. However, there are three important differences:

- ▶ Session configuration details are optional. If no configuration file is not found, the session process uses system defaults.
- ▶ All session processes read those configuration options that begin with “GEM_” and the few that are used by both Stones and Gems (such as DUMP_OPTIONS and LOG_WARNINGS). Other settings that the Gem needs are obtained from the Stone and are the same for all sessions logged in to that Stone.
- ▶ When a Gem is started on a remote node (i.e., a node other than the node on which the Stone and extents are), and there is not already a shared page cache on that node, then the configuration options (SHR_) will determine the configuration of the remote cache.

Different applications and utilities that use the same repository may need different Gem configurations. For example, batch file processes may requires a much larger amount of temporary object memory. Appendix A, “GemStone Configuration Options” describes how to specify an alternate configuration file and how to use supplementary files to adjust the system-wide configuration for a specific session process. These settings may also be passed as arguments to the Gem or to Topaz.

3.2 Configuring Gem Session Processes

In addition to configuring the Gem itself, you must determine where the Gem will be running, and ensure that any other required processes are configured correctly.

Local vs. Remote

The first step in establishing the Gem Session configuration is determining where your Gems will be running.

Local

If you intend the Gem processes or linked applications to run on the same machine as the Stone, this node needs to have sufficient resources above what is needed for the server itself.

Each Gem you plan on running requires memory as described under “Gem Memory Requirements” on page 66.

Remote

When one or more Gems will be remote (that is, running on a different node than the Stone) you will need to set the configuration that will be used on the remote node. Each remote node must be setup to run the required GemStone processes.

On a remote node, the first Gem to login will initiate the startup of the remote shared page cache, and this Gem's configuration will be used to configure the remote shared page cache.

In addition to the memory required for each Gem, as described under "Gem Memory Requirements" on page 66, you must have sufficient resources to run a remote shared page cache, cache monitor process, and other server processes. See "Server Components" on page 39 to review these requirements.

Gem Memory Requirements

The amount of memory required by a Gem session is dependent on how it is configured, as determined by the system requirements. To avoid out-of-memory conditions, Gems must be configured with an adequate temporary object cache. The default of 50MB is suitable for light use; it is likely that particular operations in an application will require more, possibly much more.

With the 50 MB default Gem's temporary object cache, the first Gem session process or linked application on a node ordinarily requires about 80 MB of memory, of which 5 MB is for code that can be shared by other session processes. Each additional session process requires about 65 MB.

If you tune the cache size for Gems, add any increase to the amount given here.

More details are provided in "Configure Temporary Object Space" on page 67, and Chapter 13, "Managing Memory", starting on page 249.

Additional Configuration for Remote Gems

The configuration file used by Gem sessions will also be used, on remote nodes, to configure the associated remote shared page cache monitor.

The Stone always creates a shared page cache monitor and cache on its own node, based on parameters in the Stone's configuration file. This cache is always used by Gems that run on this node.

On other nodes, when the first Gem session process logs in, if there is not already a shared page cache running on that remote node, a remote cache is started. Parameters specified for the first Gem that logs in determine the size of the cache and the number of processes that can attach to it (`SHR_PAGE_CACHE_SIZE_KB` and `SHR_PAGE_CACHE_NUM_PROCS`, respectively). These configuration parameter settings usually come from the Gem's configuration file, but may also be specified by Topaz arguments or arguments with the Gem's NRS.

All subsequent sessions that log in from that remote node will use the same cache.

3.3 Set the Gem Configuration Options

There are a number of configuration options for Gem session processes.

Configure Temporary Object Space

Gems use temporary object memory for a number of purposes, described in detail in Chapter 13. The upper limit on this memory is configured by the `GEM_TEMPOBJ_CACHE_SIZE` configuration option. It is important to provide sufficient temporary object space. If temporary object memory is exhausted, the Gem can encounter an out-of-memory condition and terminate.

The default of 50000 (50 MB) should be adequate for normal user sessions. For sessions that place a high demand on the temporary object cache, modifying large numbers of objects or working with large collections, a large value will be needed.

The configured value must be large enough to accommodate the memory needs of any gem using that configuration. The amount of temporary object memory required may be different for different application gems, depending on the specific tasks of the gem; you may wish to set up special configuration files for application sessions that have a particularly high demand on memory.

The full amount of `GEM_TEMPOBJ_CACHE_SIZE` is not allocated on gem login, but it is reserved and will impact memory usage per user.

The configured size of the Stone's shared page cache plus temporary object memory size should not be greater than the amount of memory available on the machine on which the Gem will be running, allowing for overhead for the operating system and other GemStone requirements. The Gem temporary object cache should also be smaller than the size of the Stone's shared page cache, otherwise it may be impossible for operations such as commit to execute.

You will probably need to experiment somewhat before you determine the optimum size of the temporary object space.

Memory management is discussed in greater detail in Chapter 13, "Managing Memory", starting on page 249.

Configure SSL for remote Gem sessions

During the RPC login process, a Secure Socket Layer (SSL) socket is used to establish the login. After that, for gem processes on the same node as their client application, communication reverts to a normal socket connection. For Gem processes that are running on a different node than their client, communication continues to use SSL.

This results in a slightly slower connection, due to the overhead of encrypting and decrypting data. To configure the gem to use normal socket communication for remote connections on secure networks, set the `GEM_RPC_USE_SSL` configuration option to false.

Private Page Cache

The setting for the `GEM_PRIVATE_PAGE_CACHE_KB` configuration option specifies the size (in KB) of the Gem's private page cache. The default value of this option is 1000; in most cases, this value is acceptable, and you do not need to do any further tuning.

When monitoring the Gem, if the value of the statistic `LocalCacheOverflowCount` is non-zero, you may want to increase the setting for `GEM_PRIVATE_PAGE_CACHE_KB` as needed.

Native Code

By default, generation of native code for Smalltalk methods is enabled. This is configured using `GEM_NATIVE_CODE_ENABLED` configuration option. When native code is disabled, execution is interpreted; behavior will be identical but somewhat slower.

Native code generation can be disabled using the runtime parameter `#GemNativeCodeEnabled`, but it cannot then be re-enabled for that session's lifetime.

If any breakpoints are set in any methods, native code is disabled. Native code is re-enabled when all breakpoints have been removed.

To determine if native code is in use by the currently executing session, execute:

```
GsProcess usingNativeCode
```

Under some configurations on x86, in particular on the Macintosh, the 32-bit offset limit may be exceeded in some cases with a very large temporary object cache. If this occurs, native code is disabled.

Note that the Foreign Function Interface (FFI) feature has additional limitations when native code is disabled. FFI is discussed in the *Programming Guide for GemStone/S 64 Bit*.

3.4 How To Access the Configuration at Run Time

GemStone provides several methods in class `System` that let you examine, and in certain cases modify, the session configuration parameters at run time.

To Access Current Settings at Run Time

`System` class methods let you examine the configuration of your current Gem session process. There are three access methods for session processes:

`gemConfigurationReport`

Returns a `SymbolDictionary` whose keys are the names of configuration file parameters, and whose values are the current settings of those parameters in the current session's Gem process.

`gemConfigurationAt: aName`

Returns the value of the specified configuration parameter from the current session, or returns `nil` if that parameter is not applicable to a session process.

`configurationAt: aName`

Returns the value of the specified configuration parameter, giving preference to the current session process if the parameter applies to a Gem.

To Change Settings at Run Time

The class method `System class >>configurationAt: aName put: aValue` lets you change the value of the internal run-time parameters in Table 3.1, provided you have the appropriate privileges.

Parameters read from the configuration file at process startup are in all uppercase, and are read-only. Parameters that can be changed at runtime have similar, but not necessarily identical, names that are in mixed case without underscores.

CAUTION

Do not change configuration parameters unless there is a clear reason for doing so. Incorrect settings can have serious adverse effects on GemStone performance. Appendix A provides additional guidance about run-time changes to specific parameters.

Table 3.1 Session Configuration Parameters Changeable at Run Time

Configuration File Option	Internal Parameter
GEM_ABORT_MAX_CRIS	#GemAbortMaxCrs
(none)	#GemConvertArrayBuilder
(none)	#GemDropCommittedExportedObjs
(none)	#GemExceptionSignalCapturesStack
GEM_FREE_FRAME_LIMIT	#GemFreeFrameLimit
GEM_FREE_PAGEIDS_CACHE	#GemFreePageIdsCache
GEM_HALT_ON_ERROR	#GemHaltOnError
GEM_KEEP_MIN_SOFTREFS	#GemKeepMinSoftRefs
GEM_NATIVE_CODE_ENABLED	#GemNativeCodeEnabled (can only be disabled at runtime)
GEM_PGSRV_COMPRESS_PAGE_TRANSFERS	#GemPgsvrCompressPageTransfers
GEM_PGSRV_UPDATE_CACHE_ON_READ	#GemPgsvrUpdateCacheOnRead
GEM_READ_AUTH_ERR_STUBS	#GemReadAuthErrStubs
GEM_REPOSITORY_IN_MEMORY	#GemRepositoryInMemory
GEM_SOFTREF_CLEANUP_PERCENT_MEM	#GemSoftRefCleanupPercentMem
GEM_TEMPOBJ_POMGEN_PRUNE_ON_VOTE	#GemPomGenPruneOnVote
GEM_TEMPOBJ_POMGEN_SCAVENGE_INTERVAL	#GemTempObjPomgenScavengeInterval

The following example changes the value of the configuration option

#GemFreeFrameLimit:

```
topaz 1> printit
System configurationAt: #GemFreeFrameLimit put: 4000
%
4000
```

For more information about the parameters that can be changed at run time, see Appendix A, "GemStone Configuration Options".

NetLDI and Interprocess Access

This chapter describes how to setup GemStone users, file permissions, and interprocess connections to ensure users have access while protecting files from unauthorized access. This chapter describes OS-level access security; GemStone login security is described in Chapter 6.

Overview (page 71)

an introduction to the GemStone processes and network objects that facilitate distributed GemStone systems.

The NetLDI (page 73)

explains GemStone's Network Long Distance Information process, and how it works within the system.

NetLDI configuration (page 74)

describes the options for configuring the NetLDI.

File Permissions (page 80)

the permissions for executable and data files work with the NetLDI modes to allow secure access for authorized users.

Linked Gem Sessions (page 82)

managing access for linked sessions that do not use the NetLDI.

4.1 Overview

As a multiuser system, GemStone must allow applications running under different UNIX userIDs to log in and perform work, while ensuring that the disk files and other resources are protected against unauthorized access or modification, inadvertent as well as intentional. There are a number of ways to configure process ownership and disk file access depending on your security requirements.

At the simplest level, with a single UNIX user on a single private node, nothing else is needed after installation. Configuring secure file and process access becomes important when you have multiple users, on a shared system, with critical or sensitive data.

In addition to OS-level process and file access, users must also authenticate with GemStone as described in Chapter 6, and object-level permissions provide additional data security as described in the *Programming Guide*. This chapter is concerned with configuring the processes and files that make up a GemStone installation.

The topics that must be considered include:

- How you will configure the NetLDI.
This may involve both the ownership and file permissions for the NetLDI executable, and the options that are specified when starting up the NetLDI.
- If user logins should also be required to provide the UNIX username and password (in addition to the GemStone UserProfile's name and password).
- The ownership, groups, and permissions for the extent files.
- The ownership and permissions for the other executables, such as the Stone and Page Servers.
- The permissions for the Shared Page Cache.

Administrative user account

Unless you are on a single-user system, or a system with no security concerns, it is recommended to create an administrative UNIX user account to install GemStone and run shared processes. This chapter assumes that you have defined such a user, and in the examples this user is *gsadmin*. The actual administrative `userId` you create is up to you.

The examples also use *gsgroup* to specify a UNIX group, that includes all UNIX users that will use GemStone.

Login Parameters and Gem process

When logging in to GemStone, client applications provide a number of *login parameters*:

- ▶ Stone name, and other details to specify the Stone.
- ▶ GemStone username and password
- ▶ Host username and password
- ▶ Gem script name (`gemnetobject`), and other specifications for the Gem.

For example, Topaz lets you set these values:

```
topaz> set gemstone nodeName
topaz> set hostusername hostUserName
topaz> set hostpassword hostPwd
topaz> set gemnetid !#netldi:netldiName!gemnetobject
```

These are all used by RPC logins; for a Linked login, only the Stone name and GemStone credentials are required.

During an RPC login, an initial request goes to the NetLDI with instructions to fork the Gem process. This request includes the host credentials and the gem script (the other login parameters are used later, by the Gem to login to the Stone).

The host username is used, along with the NetLDI configuration, to determine how to set the owner of the forked Gem process. The owner of the Gem process is important since the Gem needs to be able to open the extent files, and attach to the shared page cache, operations which should not be available to unauthorized users.

You can avoid specifying the host username and password for each login by configuring the system to run without host authentication, or by using Kerberos. While in some cases using a `.netrc` may work, the `.netrc` is inherently insecure and is not supported.

4.2 The NetLDI

GemStone NetLDIs (Network Long Distance Information) are responsible for spawning processes and providing information about GemStone processes on a given node. They are a key element connecting a distributed system together, and as such, require care to ensure that their services are available to all authorized users, but secure against unauthorized use.

In a distributed system, each node where the Stone or an RPC Gem runs must have its own NetLDI process running. NetLDIs are also required on mid-level cache nodes and to support some remote utilities.

You start a NetLDI by invoking the `startnetldi` command (described on page 353). The NetLDI, in turn, starts Gem and other processes on demand.

NetLDI Ports and Names

The NetLDI listens for all requests on a single, fixed TCP/IP port, selected either by querying the OS using `getservbyname()`, as provided by the `startnetldi -P` argument, or by randomly selecting an available unreserved port.

Client applications can then access the NetLDI either by name (either specifying the name, or relying on the default NetLDI name) or by using the port number directly.

NetLDI access by name

During the installation process, you are instructed on configuring your system to allow NetLDIs to be accessed by name. This involves adding the NetLDI name to the network services database (which can be the `/etc/services` file), and assigning a port number. The default name of the NetLDI process is `gs64ldi`. You may define your own NetLDI name instead, or in addition to `gs64ldi`, and include multiple NetLDI names with different reserved ports.

Your login parameters and command line access can then specify the NetLDI by name, for example:

```
[oboe]$ startnetldi mynetldi
topaz> set gemstone !@oboe#netldi:mynetldi!devstone
```

To access NetLDIs by name, either `gs64ldi` or your own name, the same name and port must be defined on every node, including nodes that do not need to run NetLDI; for example, client applications on Windows. This allows client nodes to reference a NetLDI by name during login.

NetLDI access by port

To avoid the need to update the network services database, you can instead access NetLDIs by port.

If `startnetldi` uses the `-P` option to specify a port, or uses a number in the port range (1024 to 65535) instead of a name, then this will be used as the listening port, provided it is not already in use. If a port is not specified in the services database, nor by the `-P` option, nor

by name, then the NetLDI will select a port at random. This port number can be used in login parameters, but since every time the NetLDI is restarted it will select a new port number and require changing the parameters, this is impractical in most cases. You can determine the NetLDI's port in this case by looking at the results of `gslist -l`.

These examples show several ways you could start the NetLDI, then use the port number in the login parameters. The differences between the `startnetldi` commands is in what name is assigned, which (if not also added to the services database), does not affect the login parameters.

```
[ofoe]$ startnetldi 10382
[ofoe]$ startnetldi -P 10382
[ofoe]$ startnetldi -P 10382 mynetldi
topaz> set gemstone !@ofoe#netldi:10382!devstone
```

Simplified access using GEMSTONE_NRS_ALL

The `GEMSTONE_NRS_ALL` environment variable provides a convenient way to set a default name or port number to access the NetLDI.

By setting this environment variable across nodes and users, access to the NetLDI from commands such as `startnetldi` and `stopnetldi`, and the parameters for login requests, will automatically use the correct NetLDI.

For example:

```
$ GEMSTONE_NRS_ALL=#netldi:mynetldi
$ export GEMSTONE_NRS_ALL
```

For more information about `GEMSTONE_NRS_ALL`, see Appendix C, "Network Resource Strings (NRS)".

4.3 NetLDI configuration

When determining the way you will configure the NetLDI, there are two decisions that need to be made: which UNIX userId will own the processes that the NetLDI forks, and what categories of requests require authentication. Each of these has several options, and the answers to the questions can combine in several ways.

There are two usual solutions:

- ▶ Authentication, with the NetLDI running as root; so Gem processes are running as the individual UNIX user who logged in.
- ▶ No authentication, with Captive Account; so all Gem processes are running as the single administrative UNIX user.

Both of these solutions provide multi-user access and appropriate security. The details of how to set up these solutions are described later, under "Setting up the NetLDI Configuration" on page 78.

The next sections go into more detail on the decisions that need to be made before you can select the right configuration.

Configuration Decisions

NetLDI behavior: who will own spawned processes?

During an RPC login, the client application passes along GemStone login parameters, including the name of the stone, the GemStone `userId`, and so on. These are sent to the NetLDI to perform the steps of the login.

These login parameters include fields for the host `UserId`, and the host password. If the host `userId` is left empty, the UNIX `userId` of the client application is used.

When the NetLDI receives the request, as part of login it forks a Gem process. The NetLDI configuration determines whether the owner of the Gem process will be the `UserId` that is provided with the login parameters, or an administrative user.

The owner of the Gem process is important since the Gem needs to be able to open the extent files, and attach to the shared page cache. While it is more intuitive to have Gem processes owned by individual `userIds`, that requires that each of these individual `userIds` have write permission to on the extent files, usually by being part of a group that has access.

The NetLDI modes are:

Default mode

In default mode, an ordinary user such as the administrative user, starts and owns the NetLDI process. Regardless of authentication level, all logins must provide the UNIX `userId` and password of the account that started the NetLDI, since a non-root user cannot fork processes that belong to another user.

This is generally suitable only for single-user systems or systems in which no additional UNIX-level authentication is required, since anyone that can login has sufficient information to access all parts of the GemStone installation.

Root mode

In Root mode, the NetLDI process is owned by the root user. This allows individual spawned processes to be owned by individual `userIds`; since root user can fork processes that will be owned by any user.

To setup root mode, the owner of the `netldid` process is changed to root, and the `setuid` bit is set. The NetLDI can be started by any user with access permission to the executable, and the NetLDI will run with an effective `UserId` of root.

Either secure or default authentication is required when the NetLDI is running in root mode. Guest authentication mode is disallowed in this mode.

Captive account

In captive account mode, all forked Gem processes are owned by a specific administrative user Id.

To specify captive account, pass the name of the designated account as an argument when starting the NetLDI, using the `-a` argument.

While technically it is possible to run the NetLDI as root and also startup in captive account, this provides no new capabilities so is not normally done.

Captive account is designed to be used with Guest mode, as described in the next section.

The effect of captive account mode is much like `setuid` method, but it only affects ownership of processes started by the NetLDI, not linked applications invoked directly by the user.

A disadvantage for some applications is that the Gem session process will perform *all* I/O as that account, not as the account running the application; including file operations and `System class >> performOnServer:.`

Also, captive account mode affects *all* services started by the NetLDI, including any ad hoc processes, which are processes started from the user's home directory. (The NetLDI looks in the user's home directory if it cannot find a service listed in `$GEMSTONE/sys/services.dat.`) If you prefer, you can prohibit such ad hoc services by specifying the `-n` option when starting the NetLDI.

Linked sessions

Linked sessions do not use a separate Gem Session; the Gem is integrated into the client process. The owner of the client application process therefore needs to have the appropriate access.

For consistency with RPC sessions started by the NetLDI, If you are running with captive account, this means using ownership and the `setuid` bit for the client application executables, to ensure client applications run as the administrative user.

Authentication: which requests need to be authenticated?

The Netldi can be configured to require authentication for all requests, for only process fork requests, or to not require authentication for any requests. Process fork requests include forking new Gems to support login; other kinds of requests includes information, such as Stone name and `gslis` results.

The authentication modes are:

Secure mode

In secure mode, all accesses, including requests for information such as `waitstone` and `gslis`, as well as process forks such as logins, require authentication. In secure mode, authentication is needed before a Gem or Stone can start a page server to access an extent or shared page cache on another node.

This mode is configured by the `-s` argument when starting the NetLDI.

In this mode, PAM (Pluggable Authentication Modules) is used to access user authentication, though the actual means of authentication, such as LDAP, depends on how your system administrators have configured your UNIX installation.

In this mode, all requests to the NetLDI must include `HostUserId` and `HostPassword`.

Fork mode

In fork mode, which is the default, only process forks require authentication. Anyone can make informational requests such as `gslis`. This is the default mode if another authentication mode is not specified.

This mode also requires PAM.

In this mode, GemStone login parameters must include `HostUserId` and `HostPassword`.

Secure or Fork mode plus Kerberos

In either Secure or Fork mode, the authentication can be done using Kerberos to bypass the need to enter the *HostUserId* and *HostPassword*.

For authentication to succeed, there must be a currently valid ticket (TGT) associated with a principle for the UNIX user id that the NetLDI is authenticating as.

Guest mode

In guest mode, no host authentication is required (GemStone login still requires authentication, of course). PAM is not needed. If userIDs other than the user that starts the NetLDI will be logging in, this is used in combination with captive account.

This is configured by the `-g` argument when starting the NetLDI. You do not need to include *HostUserId/HostPassword*; if the *HostUserId* is included, it will be used when determining ownership of the Gem process.

Table 4.1 shows how guest mode and captive account mode combinations affect NetLDI operation.

Table 4.1 NetLDI Guest and Captive Account Restrictions

startnetldi Options	Host passwords Required	Owner of Spawned Processes	Owner of NetLDI Process	Which Accounts Can Start Processes
(none)	Yes	Client's account	Ordinary user	Only the owner of the NetLDI
			Root	Any user
<code>-kkeytab</code>	No	Client's account	Ordinary user	Only the owner of the NetLDI
			Root	Any user
<code>-aaName</code> (captive account)	Yes	Account <i>aName</i> (must start the NetLDI)	Ordinary user (<i>aName</i>)	Only <i>aName</i>
			Root	Any user
<code>-g</code> (guest mode)	No	Client's account	Ordinary user	Only the owner of the NetLDI
			Root – not allowed	
<code>-aaName -g</code> (guest mode with captive account)	No	Account <i>aName</i> (must start the NetLDI)	Ordinary user (<i>aName</i>)	Any user
			Root – not allowed	

Setting up the NetLDI Configuration

As described in the previous sections, there are two common combinations of features that configure the NetLDI for secure multi-user accessibility.

1. Setting up NetLDI as root with Authentication

To run the NetLDI with an effective UserId of root, you can change the ownership of the NetLDI executable, and set the S bit (setuid). This allows any user with execute permissions to start the NetLDI and let that NetLDI run as root.

If you are logged in as root when you run the GemStone installation program, it offers to set the ownership and permissions for the NetLDI. To set them manually, do the following as root:

```
# cd $GEMSTONE/sys
# chown root netldid
# chmod u+s netldid
```

The resulting file permissions and ownership for the NetLDI executable should look similar to this:

```
-r-sr-xr-x 1 root gsgroup 2007036 Mar 7 11:29 netldid
```

In this mode, PAM (Pluggable Authentication Modules) is used to authenticate, and the system should either configure a service with the name `gemstone.netldi`, or ensure that the default PAM authentication will allow logins. The actual means of authentication, such as LDAP, depends on how your system administrators have configured your UNIX installation.

Starting the NetLDI

When you start the NetLDI, you may choose to authenticate all requests by including the `-s` argument to `startnetldi`, or omit that argument and only authenticate requests for process forks. Using the `-g` for guest mode is not allowed with the NetLDI running as root.

Login parameters

Your application's login interface will generally let you specify a UNIX login name and password for the node on which you will be running an RPC Gem session process. For example, Topaz lets you set these values:

```
topaz> set hostusername HostUserName
topaz> set hostpassword HostPwd
```

GemBuilder for Smalltalk (GBS) provides similar fields in its login dialog, as does GsExternalSession, and other interfaces provide ways to enter this information.

NOTE

Authentication is always done using the "real" user id, not the effective user id as set by the S bit on GemStone executables.

These fields are used to compose instructions in NRS syntax to be sent to the NetLDI. For example, if you set the Topaz login parameters `HostUserName` and `HostPwd`, the application puts them in an NRS like the following:

```
'!@Server#auth:HostUserName@HostPwd!gemnetobject'
```

Although it is less convenient for ordinary use, this can be done manually, by entering the authorization modifier directly using the Topaz *GemNetId* parameter. For example:

```
topaz> set gemnetid !@Server#auth:HostUserName@HostPwd!gemnetobject
```

When NRS is used to login using *GsExternalSession*, you can compose the NRS programmatically using the kernel class *GsNetworkResourceString*. See the *Programming Guide* for more on External Sessions, and the classes in the image.

Using Kerberos authentication

The NetLDI can be configured to use Kerberos for authentication for the *hostUserId*/*hostPassword*.

To configure Kerberos authentication for NetLDI:

- ▶ **startnetldi** must be executed with the **-k** option pointing to the Kerberos keytab file.
- ▶ The GemStone login parameters must have an empty *hostPassword* field.
- ▶ There must be a currently valid ticket (TGT) associated with a principle for the UNIX user id that the NetLDI is authenticating as.

The **-k** option cannot be used with guest mode; no authentication is done in guest mode.

2. Setting up NetLDI in Guest Mode with Captive Account

To configure guest mode with captive account, do the following

If you do not already have one, create an OS account to own the GemStone distribution tree and serve as the captive account. This will be referred to as *gsadmin*.

1. Make *gsadmin* the owner of the distribution tree
2. Set the *setuid* bit for any linked GemStone executables that run on the server node.
3. Make the repository extents accessible only by *gsadmin* (mode 600). For instructions, see “How To Set Up a Raw Partition” on page 52.
4. Make sure that *gsadmin* has execute permission for `$GEMSTONE/sys/netldid`.

The *setgid* bit should NOT be set on the *netldid* executable; it should look similar to this:

```
-r-xr-xr-x 1 gsadmin gsadmin 674488 Mar 7 11:29 netldid
```

Starting the netldi

Log in as the captive account (such as *gsadmin*).

Start the NetLDI using the arguments for guest mode and captive account. For instance:

```
% startnetldi -g -a gsadmin
```

You may specify other *startnetldi* options.

Login

You do not need to specify the *hostUserId* and *hostPassword* parameters.

4.4 File Permissions

The considerations in setting file permissions is the same as in configuring the NetLDI; to protect the repository extents, while ensuring that authorized users can use the repository.

All reads and writes to the extents should be done through GemStone repository executables: the executables that run the Stone, Page Servers, and Gems.

This does not include file copy backups of the GemStone extents, nor restore if that is needed. These are OS level operations that also require protection.

For the tightest security, you can have the extents and executables owned by a single UNIX account, using the setuid bit on the executable files, and making the extents writable only by that account. This way, all processes started from any of the executables are owned by a single user, and only that user can write the extent files.

While most processes that write files such as fileouts and programmatic backups will set the ownership of the resulting file to the client user, this may not be possible for linked sessions in client applications, if the client application does not distinguish between real `userId` and effective `userId`.

If this is an issue, rather than using the administrative account, you can make the extents writable by a particular UNIX group and have all users belong to that group. While this means all Gems are owned by the individual user accounts, there is the risk of inadvertent damage or deletion of the extent files.

Using the Setuid Bit

When all extents and executables are owned and can only be written by a single UNIX account, it provides the strongest security. By setting the setuid bit, the processes started from that executable are owned by the owner you specify for the file.

Table 4.2 shows the recommended file settings. In this table, *gsadmin* and *gsgroup* can be any ordinary UNIX account and group (do NOT use the root account for this purpose). The person who starts the Stone must be logged in as *gsadmin* or have execute permission.

Table 4.2 Recommended File and Executable Permissions for the Server

Resource or Process	Filename	Protection Mode	File Owner	File Group	Process Runs As
Repository extents	(default) data/extent*.dbf	-rw-----	gsadmin	gsgroup	
Stone	sys/stoned	-r-sr-xr-x	gsadmin	gsgroup	gsadmin
AIO and Free Frame Page Servers	sys/pgsvrmain	-r-sr-xr-x	gsadmin	gsgroup	gsadmin
Gem	sys/gem	-r-sr-xr-x	gsadmin	gsgroup	gsadmin

Ownership and permissions for the `netldid` executable depend on the authentication mode chosen, and are discussed in Chapter 3.

If you are logged in as root when you run the GemStone installation program, it offers to set file protections in the manner described in Table 4.2. To set them manually, do the following as root:

```
# cd $GEMSTONE/sys
# chown gsadmin gem pgsvr pgsvrmain stoned
# chmod u+s gem pgsvr pgsvrmain stoned
# cd $GEMSTONE/data
# chown gsadmin extent0.dbf
# chmod 600 extent0.dbf
```

You must take similar steps to provide access for repository linked clients, which is described under “Linked Gem Sessions” on page 82.

Alternative: Use Group Write Permission

For sites that prefer not to use the setuid bit, the alternative is to make the extents writable by a particular UNIX group and have all users belong to that group. That group must be the primary group of the person who starts the Stone (that is, the one listed in `/etc/passwd`). Do the following, where *gsgroup* is a group of your choice:

```
% cd $GEMSTONE/data
% chmod 660 extent0.dbf
% chgrp gsgroup extent0.dbf
```

Sites that run linked sessions may also prefer to use this protection so that fileouts and other I/O operations that do not read or write the repository will be done using the individual user’s id instead of the single *gsadmin* account.

Shared Page Cache

The shared memory and semaphore resources used by GemStone are created and owned by the user account under which the Stone repository monitor is running and have the same group membership. Access for the shared page cache is set by the `SHR_PAGE_CACHE_PERMISSIONS` configuration option; by default, it is read-write for the owner and read for the group (the equivalent of file protection 640). You can inspect the cache ownership and permissions by using the `ipcs` command.

For a session to log in using a shared page cache, the OS user account of the linked application or Gem session process must either be the same as that of the Stone (such as the *gsadmin* account) or be one that belongs to the same group as the Stone. The same requirement applies to page server processes, which are discussed in Chapter 5, “Connecting Distributed Systems”.

If the setuid bit is set on repository executables, the Stone process and shared page cache will belong to the owner you specify for those files (such as *gsadmin*).

File Permissions for Other Files and Directories

GemStone creates log files and other special files in several locations. In a multi-user environment, the protection of these resources must be such that the appropriate file can be created or updated in response to actions by several users.

```
/opt/gemstone/locks
```

All users should be able to read files in the directory `/opt/gemstone/locks` on each node (or an equivalent location, as discussed on page 31).

```
/opt/gemstone/log
```

Users who will start a NetLDI process require read, write and execute access to `/opt/gemstone/log`, if this directory is used for logging.

```
system.conf
```

The user who owns the Stone process must have write permission to the configuration file, since the Stone must be able to write as well as read its primary configuration file. Writes are made if an extent is added while the Stone is running, so that subsequent restart will be correct. By default, this file is `$GEMSTONE/data/system.conf`.

```
$HOME
```

GemStone by default creates log files for session processes, such as RPC Gems, in the home directory of the user owning the Gem. This may be the owner of the Gem executable, or the user specified for the NetLDI's captive account.

An alternate location can be specified using NRS syntax. This NRS can be included in the login parameters, or defined by `GEMSTONE_NRS_ALL`. See "Controlling log file names and locations" on page 371.

4.5 Linked Gem Sessions

GemStone supports both RPC logins, in which the Gem is a separate process from the client application, and linked logins, in which the Gem is in the client application process.

The above discussions of executable permissions are focused on the issues with RPC logins, which are facilitated by the NetLDI. For linked logins, however, the client application, which could be topaz, client Smalltalk such as VisualWorks or VA/Smalltalk, Pharo, GemBuilder for Java, a GCI Application, or something else, is also involved.

For linked applications *on the server*, we recommend you use the `setuid` bit on the application's executable file, and have the file owned by an administrative user, *gsadmin*. This works well for linked topaz logins. The `installgs` script offers to set the file ownership and permissions for you. To do it manually, do this (you may need to login or `sudo` as root):

```
# cd $GEMSTONE/bin
# chmod u+s topaz
# chown gsadmin topaz
```

GemStone's topaz executable performs repository reads and writes as the *effective* user (the account that owns the executable's file), but performs other reads and writes as the *real* user (the one who invoked it).

For linked applications that do not distinguish between real and effective user IDs, you may prefer *not* to use the `setuid` bit. Linked applications that do not make this distinction are likely to perform *all* I/O as the effective user, or *gsadmin*. In this case, it may be preferable to remove the `S` bit on that executable and add group write permission to the extents.

Connecting Distributed Systems

This chapter tells how to set up GemStone/S 64 Bit in a distributed environment:

Overview (page 83)

An introduction to the GemStone processes and network objects that facilitate distributed GemStone systems.

Configuring GemStone on Remote Nodes (page 88)

Examples for setting up typical distributed client-server configurations.

Troubleshooting Remote Logins (page 96)

What to do when there are problems with remote logins.

Tuning for Distributed Configurations (page 101)

Tuning interprocess communications in a distributed system.

5.1 Overview

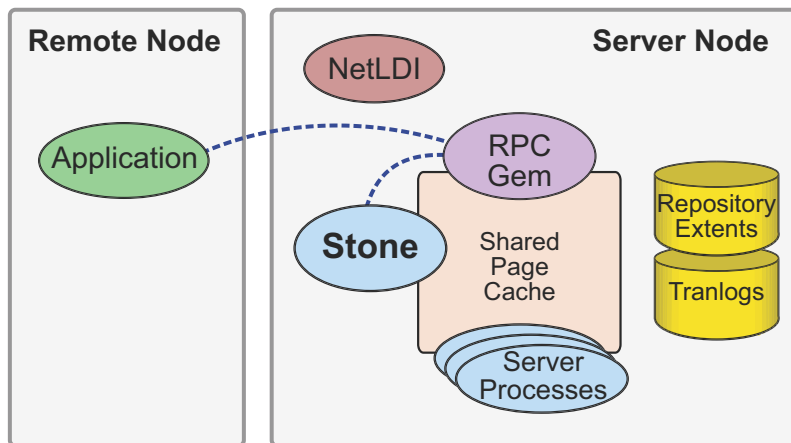
GemStone is designed to be highly efficient and easy to use in a networked environment with many nodes. However, this requires additional steps by the system administrator so that the server processes can be found and accessed, and so server processes can connect to one another. Because processes are running on different nodes, the log files are spread throughout the network, and troubleshooting may become more complicated.

Figure 5.1 and Figure 5.2 show two typical distributed configurations in which an application on a remote node is logged in to a repository and Stone repository monitor running on a server node.

Distributed configuration with Local Gem

In Figure 5.1, an application communicates with a Gem session process on the server node by way of RPC calls. This configuration lets the Gem execute Smalltalk code in the repository without first bringing complex objects across the network. The Gem directly accesses the shared page cache that was started by the Stone repository monitor. This configuration is considered a "Local" Gem, since the Gem is local to the Stone.

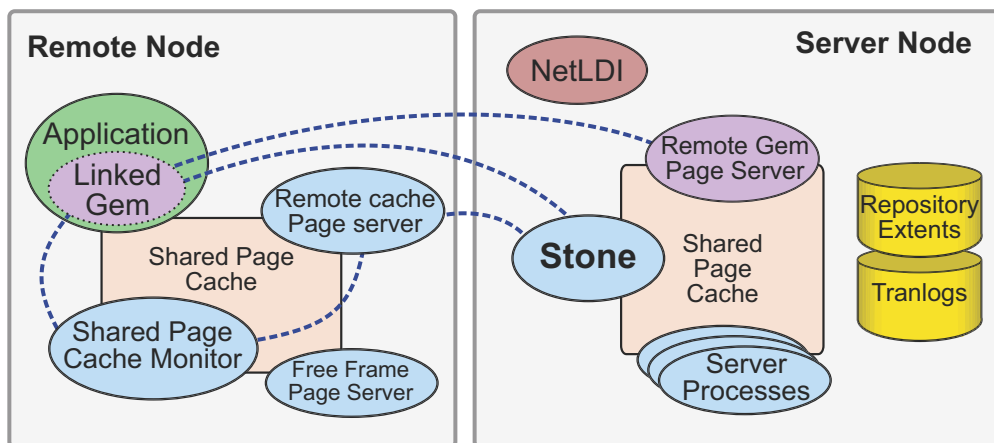
Figure 5.1 Gem Session on Server Node



Distributed configuration with Remote Gem

In Figure 5.2, the application and the Gem are linked in a single process that runs on the remote node (i.e, not the Stone's node). This configuration avoids the overhead of RPC calls, but may increase network traffic if large objects must be brought across the network. When a Gem is on a remote node, it needs a shared page cache on that node. This is started by the first Gem on that node. Page server processes are also needed to handle pages between the two nodes.

Figure 5.2 Gem Session on Remote Node



Network

The nodes in your system can be any combination of GemStone-supported platforms, as long as they have a network connection. Inter-node communication is handled via sockets.

Nodes in which the Stone, or a Gem process, will run must be platforms supported by GemStone/S 64 Bit.

Client-process only nodes such as the remote node in Figure 5.1 can include supported client platforms, such as Windows, as well as server platforms.

GemStone NetLDIs

The NetLDIs are the glue holding a distributed GemStone system together. Each NetLDI spawns other GemStone processes on request, and reports the location of GemStone services on its node to remote processes that must connect to those services.

In a distributed system, each node where a Stone repository monitor or Gem session process runs must have its own NetLDI process running. However, nodes that run only linked applications, or client applications without a Gem on that node, do not require a NetLDI.

The NetLDI listens on a well-known port for requests for RPC logins or other services.

NRS Syntax

GemStone uses Network Resource Strings (NRS) to specify the name and location of each part of the GemStone system. NRS is widely used in utility commands and in login parameters, to specify the location and name of the Stone and NetLDI, and where the other GemStone processes should run.

The most useful parts to know are:

`!@node#netldi:netldi!stoneOrGemService`

- ▶ The “@node” can be omitted for the local node, and you may use an IP address instead of the name of the node.
- ▶ the “#netldi:netldi” can be omitted if the netldi is running with the default name `gs64ldi` and that name is recorded in the services database. The netldi can be specified by name or by port number.
- ▶ `stoneOrGemService` is the name of the running Stone, or `gemnetobject` or another Gem service. This part is required; if you omit both node and netldi, you do not need the ! dividers; you can use simply `stoneOrGemService`.

Appendix C, “Network Resource Strings (NRS)” describes NRS in detail.

In particular, you may find it very useful to set the `GEMSTONE_NRS_ALL` environment variable if you are using a NetLDI name other than the default.

Stone

When logging into to a Stone on the local machine, the stone name is sufficient. If the Stone is on a separate node, you can specify the location of the stone, and the name of the NetLDI on the Stone's node, using NRS.

For example, to connect to a stone named `devstone` on the node `oboe`, with the default-named NetLDI `gs64ldi`:

```
topaz> set gemstone !@oboe!devstone
```

Connecting to the RPC Gem

When a client requests an RPC login, a Gem session process needs to be spawned to fulfill the login request. To do this, the login parameters include the gem service (in topaz, the `gemnetid`), usually `gemnetobject`, `gemnetdebug`, or the name of a custom gem script.

For example,

```
topaz> set gemnetid gemnetobject
```

Using `gemnetobject` alone as here, specifies to run the gem on the same node as the client application.

By specifying an NRS, in the gem service login parameter that includes the node and the NetLDI on which you want the Gem to run, you can instruct GemStone on which processes need to be started and where. This can be the same node as the Stone. It can also be on another node entirely, provided GemStone is installed and a NetLDI is running on that node.

For example, if you intend the gem to run on a machine named `lark`, with the default-named NetLDI `gs64ldi`:

```
topaz> set gemnetid !@lark!gemnetobject
```

Note that if the gem service NRS includes a node name and the stone NRS does not include a node name, then the Stone is expected to be on the node in the Gem NRS, not on the client node.

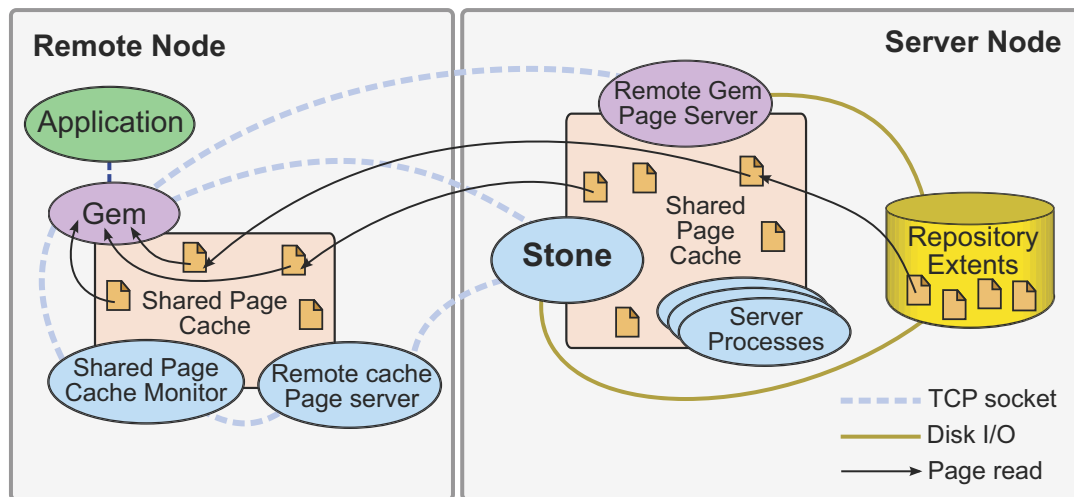
Shared Page Cache

There is always a shared page cache on the Stone's node. Any Gems that are on the Stone's node (local Gems) will use that shared page cache.

When a GemStone login specified a Gem run on a remote node, then a shared page cache and the associated processes, shared page cache monitor and free frame page server, need to be started on that node. Normally, the cache is started up by a NetLDI running on the remote node, although a linked session (but not an RPC session) can also initiate the startup.

When the remote Gem wants to access a page in the repository, it first checks the shared page cache on its own node. If the page is not found, the Gem communicates with the Remote Gem Page Server on the server node, checking in the shared cache on the server node and if necessary, reading the page from the disk.

Figure 5.3 Shared Page Cache with Remote Gem



GemStone Page Servers

GemStone uses specialized page servers for various purposes. As described in Chapter 1, the AIO page server carries out GemStone repository I/O, and the Free Frame page server adds free frames to the free frame list.

In addition, Gems on remote nodes require page servers:

- ▶ When a Gem connects to a repository extent across the network, it requires a Remote Gem Page Server on the Stone's node, associated with the remote node. This is spawned if it is not already running. This page server looks up pages in the Stone's cache and performs extent reads, on behalf of the gem
- ▶ On the remote node, a Remote Cache Page Server supports the Stone in managing the remote shared page cache.

Port use in GemStone

GemStone's interprocess and internode communications are established via socket connections. A number of key GemStone processes listen on well known ports for a connection, which initiates the handshakes that establish interprocess socket connections.

Processes that listen on well known ports include the NetLDI, the Stone, and the Shared Page Cache Monitor. These named GemStone processes create a *serviceName.LCK* at startup, in the directory `/opt/gemstone/locks` or another location as described on page 31. This file encodes the well-known port that the process is listening on. When the NetLDI (for example) contacts a particular Stone, it first looks up the well-known port for that Stone by locating the lock file for that Stone, then contacts the Stone on that port.

The NetLDI's port is specified on startup in a number of ways, as described on page 73.

The ports used by the Stone and Shared Page Cache Monitor can be specified using `STN_WELL_KNOWN_PORT_NUMBER` and `SHR_WELL_KNOWN_PORT_NUMBER` configuration options, respectively. These must be valid port numbers that are not already in use.

Disrupted Communications

Several incidents can disrupt communications between the GemStone server and remote processes in a distributed configuration. This can include network glitches, network overload, crashes or unexpected shut down of a remote process or the entire remote node, or of the network channel itself.

GemStone ordinarily depends on the network protocol *keepalive* option to detect that a remote process no longer exists because of an unexpected event. The keepalive interval is set globally by the operating system, typically at two hours. When that interval expires, the GemStone process tries to obtain a response from its partner. The parameters governing these attempts also are set by the operating system, with up to 10 attempts in 15 minutes being typical. If no response is received, the local GemStone process acts as if its partner was terminated abnormally. The keepalive interval may need to be adjusted.

Firewalls between GemStone processes require that the communications ports be configured to be open.

5.2 Configuring GemStone on Remote Nodes

If you are only running local Gems; in other words, only the client application will run on the remote node, most of the configuration for the Gem is on the server node. However, if you will run Gems or Linked sessions on nodes remote from the Stone, then more configuration is required on the remote node.

Local Gems only

If you are only running Gems on the same node as the Stone, there are only a few steps required on the remote node.

- ▶ Install the client shared libraries. This can be done by a complete server installation, or by copying the small number of required library files to the client, and configuring your client search paths appropriately.

Details on how to do this, and the various options, are in the *GemStone/S 64 Bit Installation Guide* for the client platform.

- ▶ If you are using named NetLDIs, you must configure your client system services database to map the same NetLDI name to the same port as defined on the server. If you are using a local services database, edit `/etc/services` or, on Windows, `\WINDOWS\system32\drivers\etc\services` to add the appropriate entry.

Remote Gems

Configuring GemStone so that Gem sessions or linked applications can run on the remote node is much the same as configuring the Gem and cache a session process on the server, which is described in Chapter 3, “Configuring Gem Session Processes”.

The remote node will need access to a number of executables and shared libraries within the server installation. You can either repeat the installation from the GemStone distribution media, or mount the directory on the server node that contains `$GEMSTONE`.

If you repeat the installation on the remote node, we recommend that you also run `$GEMSTONE/install/installgs`. In particular, you should make the same selections

regarding the ownership and group for the GemStone files as you did on the primary server node. You can save disk space later by deleting initial repositories (`$GEMSTONE/data/extent0.dbf` and `$GEMSTONE/bin/*.dbf`) and the complete upgrade (`$GEMSTONE/upgrade`) and seaside (`$GEMSTONE/seaside`) directories.

Some additional details points to keep in mind:

- ▶ A remote node (on which a Gem is running) must have its kernel configured for shared memory similarly to how it is configured on the primary server node.
- ▶ Only nodes running a Stone need a GemStone key file, not nodes running remote sessions.
- ▶ It's best if each node has its own directory for `/opt/gemstone/log` and `/opt/gemstone/locks` (or `/usr/gemstone/log` and `/usr/gemstone/locks`, or other location under `$GEMSTONE_GLOBAL_DIR`). If these directories are on an NFS-mounted partition, make sure that two nodes are not using the same directories. Each Stone and NetLDI needs a unique lock file. Shared log files may make it impossible to diagnose problems.
- ▶ If you are using named NetLDIs, you must configure your client system services database to map the same NetLDI name to the same port as defined on the server. If you are using a local services database, edit `/etc/services` or, on Windows, `\WINDOWS\system32\drivers\etc\services` to add the appropriate entry.
- ▶ Unless you run the NetLDIs in guest mode with a captive account, users must have an account on, and authorized network access to, all nodes that are part of the GemStone network for the repository they will be using. This includes the nodes on which the Stone Repository monitor and the user's Gem session process reside.

Unless you run the NetLDIs in guest mode with a captive account, *the user who starts the Stone repository monitor* ordinarily needs an account on *every* node where a Gem session process will run.

Configuration Examples

GemStone supports several configurations in which the application communicates with the Gem session process by using remote procedure calls (RPCs), as well as linked applications. This section presents examples that illustrate the processes and interprocess connections within the following distributed applications:

- ▶ RPC Application on a Remote Node with Local Gem (page 90)
- ▶ Linked Application on a Remote Node (page 91)
- ▶ RPC Application on a Remote Node with Remote Gem (page 92)
- ▶ RPC Application, Gem, and Stone on Three Nodes (page 93)
- ▶ Distributed System with a Mid-Level Cache (page 94)

Determine if you will run RPC or Linked application

GemStone shared libraries are provided in two variants: linked and RPC. In order to login a linked session, you must be running an application bound to the linked libraries, such as `topaz -l`. To login an RPC session, you may run an application bound with the RPC libraries or with the linked applications.

A linked application may have only one linked login, but may login RPC sessions in addition to (or instead of) the linked login. RPC sessions applications are not able to login linked sessions.

Set the environment

The instructions assume that you are already set up to run GemStone applications, as described in Chapter 3. In particular, you must have defined the GEMSTONE environment variable and invoked `$GEMSTONE/bin/gemsetup.sh` or its equivalent, so the system search path include `$GEMSTONE/bin`.

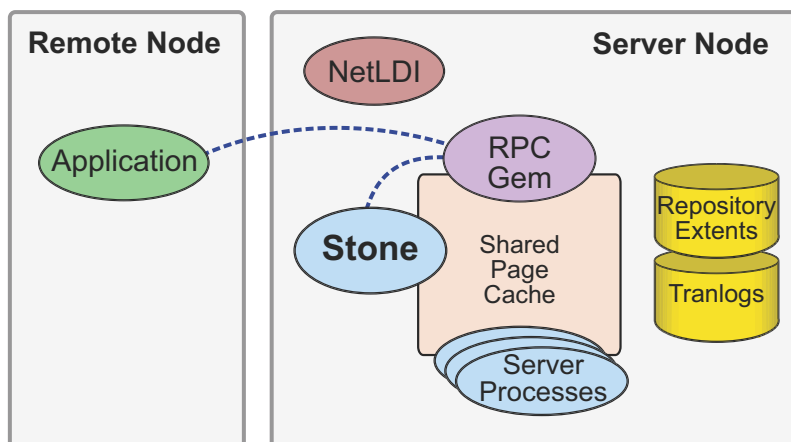
Configure NetLDI and authentication mechanism

For all these examples, you must have a NetLDI running on the Stone's host. In many cases you will also need a NetLDI on the remote node or nodes. These should be configured according to your security requirements, as described under "NetLDI configuration" on page 74.

RPC Application on a Remote Node with Local Gem

If the Gem session process is going to run on the server node (as shown in Figure 5.4), an RPC application uses a NetLDI on that node to start a Gem session process. Unless the NetLDI is running in guest mode with a captive account, the application user must provide authentication to the NetLDI. The login parameters include the specification for the Gem network object (`gemnetobject`). For more information about network objects and how to invoke them, see "Connecting to the RPC Gem" on page 86.

Figure 5.4 RPC Application on a Remote Node with remote Gem



To login an RPC session with the Gem on the Stone's host, set the login parameters as follows:

- Set the Stone's name:

```
topaz> set gemstone stoneName
```

The hostname is optional in this case; by default, it will use the host specified by the NRS in the gem service parameter (`gemnetid`)

- Set GemStone user name and password:

```
topaz> set username DataCurator
topaz> set password swordfish
```

You may omit the password; you will be prompted for it on login.

- Set host authentication, if your NetLDI is not running in guest mode:

```
topaz> set hostusername unixUserAccountName
topaz> set hostpassword passwordForUnixUserAccount
```

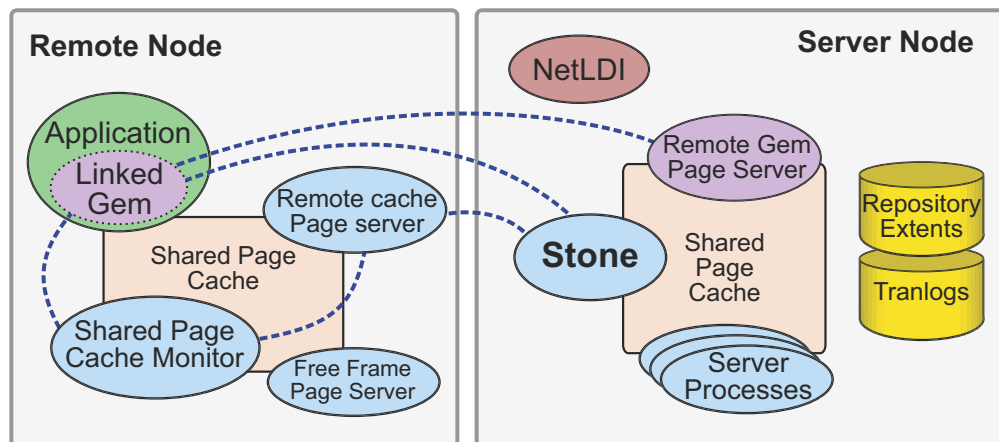
- Set the gem service, specifying the stone's host name, and the name of the NetLDI if necessary. For example, either one of the following:

```
topaz> set gemnetid !@serverhost!gemnetobject
topaz> set gemnetid !@serverhost!netldi:NetLdiName!gemnetobject
```

Linked Application on a Remote Node

Figure 5.5 shows how a linked application on a remote node communicates with a Stone and repository on the primary server node.

Figure 5.5 Connecting a Linked Application to a Remote Server



In this configuration, a remote, subordinate shared page cache is started on the remote node, along with associated server processes. The configuration of this remote cache is set by the first session to log in on this node. After the last session on this node logs out, the cache will shut down.

In addition, this configuration requires a Remote Gem page server on the server node, which reads pages from the repository on behalf of the gem.

Because the shared page cache is readable and writable only by its owner and members of the same group (protection 660), the user running the application may need to belong to that group. See "Shared Page Cache" on page 81.

Set the login parameters as follows:

- Set the Stone's name, one of the following:


```
topaz> set gemstone !@serverhost!stoneName
topaz> set gemstone !@serverhost!netldi:NetLdiName!stoneName
```
- Set GemStone user name and password.


```
topaz> set username DataCurator
topaz> set password swordfish
```

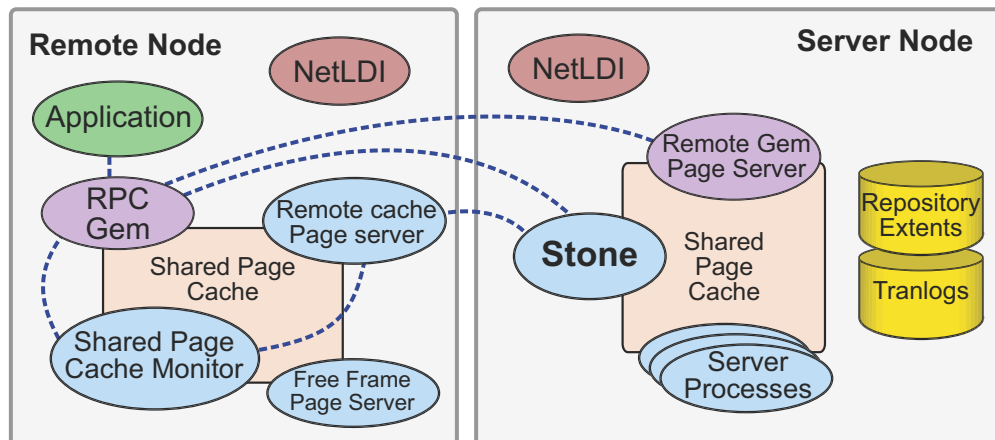
You may omit the password; you will be prompted for it on login.
- Set host authentication, if your NetLDI is not running in guest mode:


```
topaz> set hostusername unixUserAccountName
topaz> set hostpassword passwordForUnixUserAccount
```
- Do **not** set the gem service; doing so specifies that the login will be RPC, rather than linked.

RPC Application on a Remote Node with Remote Gem

The configuration shown in Figure 5.6 shows the RPC application and its session process running on the same node. This configuration is similar to a linked application, but in this case, a NetLDI is required on the remote node in order to launch the RPC Gem session process.

Figure 5.6 Starting the Session Process on a Remote Node



- Set the Stone's name, one of the following:


```
topaz> set gemstone !@serverhost!stoneName
topaz> set gemstone !@serverhost!netldi:NetLdiName!stoneName
```

You must specify the Stone's host, since the otherwise it will assume the Stone is on the same node as the Gem, given in the gemnetid's NRS.
- Set GemStone user name and password.


```
topaz> set username DataCurator
topaz> set password swordfish
```

You may omit the password; you will be prompted for it on login.

- Set host authentication, if your NetLDI is not running in guest mode.

```
topaz> set hostusername unixUserName
topaz> set hostpassword passwordForUnixUserAccount
```

- Set the gem service, specifying the stone's host name, and the name of the NetLDI if necessary. For example, one of the following:

```
topaz> set gemnetid !@remotemode!gemnetobject
topaz> set gemnetid !@remotemode!netldi:NetLdiName!gemnetobject
```

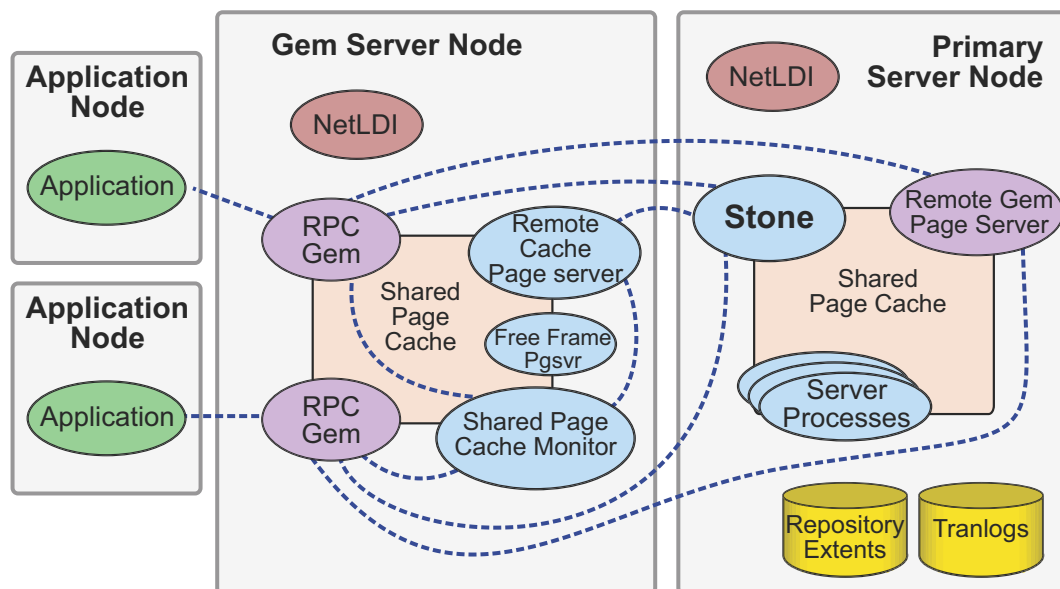
RPC Application, Gem, and Stone on Three Nodes

The RPC application, session process, and Stone can run on three separate nodes. The application runs on its node and connects to a Gem session process on the Gem's node, which is sometimes referred to as the Gem Server. That session process communicates with the repository on the primary server node by way of a page server.

Again we see that a NetLDI must be running on each node where part of GemStone executes (but is not required on the application node, which runs only the RPC application).

The following diagram shows two separate RPC applications connecting to Gems running on the same node, the Gem Server node.

Figure 5.7 RPC Applications using a Gem Server Node



The directions for this configuration are the same as for the previous configuration, "RPC Application on a Remote Node with Remote Gem"; however, the application itself is executing on a third node.

- Set the Stone's name, one of the following:

```
topaz> set gemstone !@serverhost!stoneName
topaz> set gemstone !@serverhost!netldi:NetLdiName!stoneName
```

You must specify the Stone's host, since the otherwise it will assume the Stone is on the same node as the Gem, given in the gemnetid's NRS.

- Set GemStone user name and password.

```
topaz> set username DataCurator
topaz> set password swordfish
```

You may omit the password; you will be prompted for it on login.

- Set host authentication, if your NetLDI is not running in guest mode.

```
topaz> set hostusername unixUserAccountName
topaz> set hostpassword passwordForUnixUserAccount
```

- Set the gem service, specifying the stone's host name, and the name of the NetLDI if necessary. For example, one of the following:

```
topaz> set gemnetid !@remotenode!gemnetobject
topaz> set gemnetid !@remotenode!netldi:NetLdiName!gemnetobject
```

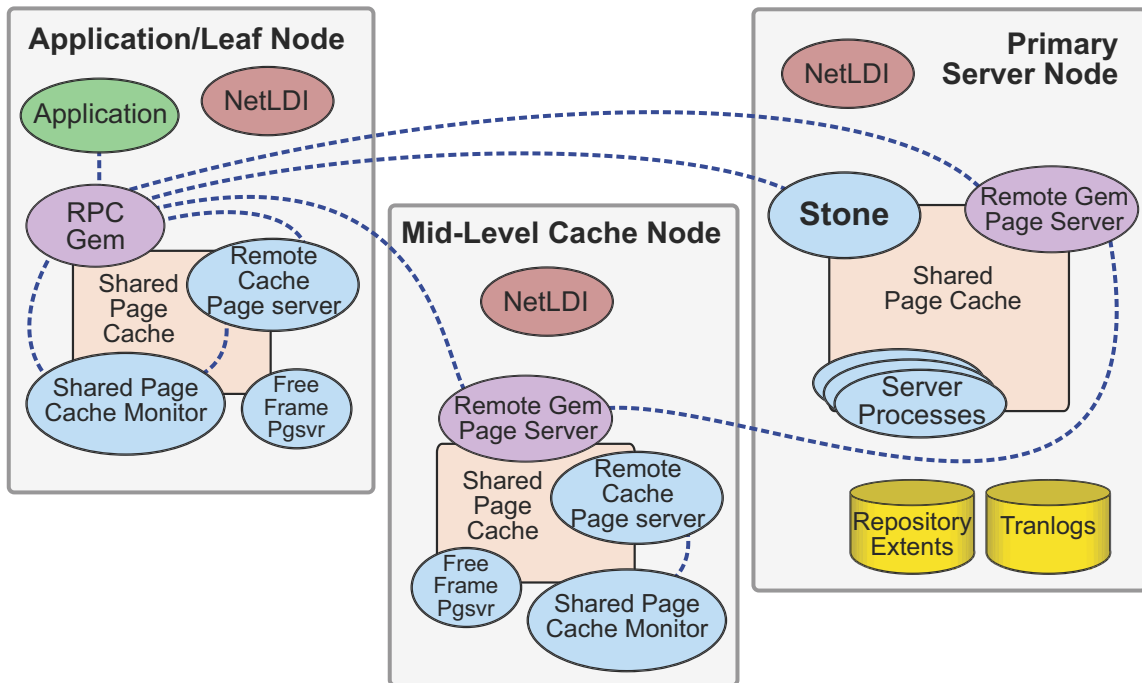
Distributed System with a Mid-Level Cache

In a distributed system over a Wide Area Network (WAN), with many remote nodes that are topographically distant from the Stone but close to each other, a mid-level cache can improve performance for the remote sessions. In this configuration, the RPC application and the session process may be on the same or different nodes, with the mid-level cache and Stone running on separate nodes. In Figure 3.8, the Gem session process (on the leaf node) connects to a mid-level cache (on the mid-level cache node). The session process communicates with the repository (on the primary server node) by way of a page server. When the Gem needs a page but can't find it in its local cache, it first looks in the mid-level cache. If the Gem can't find the page in the mid-level cache, it then forwards the request to the page server on the Stone's host.

The Stone's page server aggregates the responses from the page servers on each of the Gem's shared caches, and returns a combined response to the Stone. This reduces the number of round trips from the Stone to distant nodes.

If a mid-level cache is in use, then for each Gem process using the mid-level cache, all the shared caches to which the Gems are attached are subordinate to that mid-level cache.

Figure 5.8 An Application with a Mid-Level Cache



Setting up a configuration with a mid-level cache requires that the session execute code following login to start a mid-level cache on a specified host, or to connect to an existing mid-level cache. Unlike the other remote configurations discussed in this chapter, the configuration is not established entirely by configuration settings and login arguments.

The Gem must be on a node that is remote from the Stone, and the request to connect to a mid-level cache must specify a node that is neither the Stone's nor the Gem's node.

If a Gem is running on the same machine as a mid-level cache, that Gem will use the mid-level cache as its local cache.

Connection Methods

System Class methods in the Shared Cache Management category allow you to connect to a mid-level cache.

```
midLevelCacheConnect: hostName
```

Attempts to connect to a mid-level cache on the specified host, if the cache already exists. The session's Gem process must be on a machine different from the machine running the Stone process.

```
midLevelCacheConnect: hostName cacheSizeKB: aSize  
maxSessions: nSess
```

If a mid-level cache does not already exist on the specified host, and *aSize* > 0, attempts to start the cache and connect to it. If a cache is already running on the host, this method attempts to connect to the cache and ignores the other arguments.

The size of the mid-level cache is controlled by the method argument *aSize*, rather than by configuration parameters (as with other shared caches).

For example,

```
topaz> set gemstone !@Server!gs64stone
topaz> set username DataCurator
topaz> set gemnetid !@gemNode!gemnetobject
topaz> login
GemStone Password?
<details omitted>
successful login
topaz 1>
topaz 1> printit
    System midLevelCacheConnect: 'midLevelCacheNode'
        cacheSizeKB: 2048 maxSessions: 10
%
```

Reporting Methods

System Class methods in the Shared Cache Management category return lists of the shared caches on your system.

`remoteCachesReport`

Returns a String that lists all shared caches that the Stone process is managing, not including the cache on the Stone machine.

`midLevelCachesReport`

Similar to `remoteCachesReport`, but only includes the mid-level caches.

5.3 Troubleshooting Remote Logins

Logging into GemStone from a remote node requires proper system configuration of the remote node and frequently requires permission for network access from the primary server to the remote node as well as from the remote node to the primary server.

- ▶ The UNIX kernel on the remote node should meet shared memory and semaphore requirements similar to those for the server, although smaller sizes may be sufficient.
- ▶ Make sure that NetLDIs are running on all nodes that require them (all nodes except remote nodes that do not run RPC Gems), and that the NetLDIs are the same version as your environment, and that the `startnetldi` that started the NetLDI processes used the appropriate options to specify authentication (guest mode, or Kerberos, for example).
- ▶ Make sure that all named NetLDIs have the same port number in the `services` database/s for each node, and that all node names are listed in `/etc/hosts`.
- ▶ If an RPC application is being started (that is, one with a separate Gem session process), check that the authentication is valid; the user has entered their username and password, if needed, and if Kerberos is used, that their ticket is valid. See “Setting up the NetLDI Configuration” on page 78 for NetLDI authentication.

- ▶ Ownership and permissions for `$GEMSTONE/sys/netldid` must be appropriate for the authentication system in use (details can be found under “Setting up the NetLDI Configuration” on page 78
- ▶ the directories in `/opt/gemstone` must be writable.
- ▶ Ensure that the owner of the Gem process has an account on the node where the Gem will run and needs access to the Gem log location, typically in `$HOME`; and that the user who started the Stone has an account on the remote node. This user also may require write permission for `$HOME`, if the remote server process log files are not configured to write to a different location. .
- ▶ Check any `GEMSTONE` environment variables for definitions that point to a previous version: `env | grep GEM`.

How the Login Process starts Session Processes

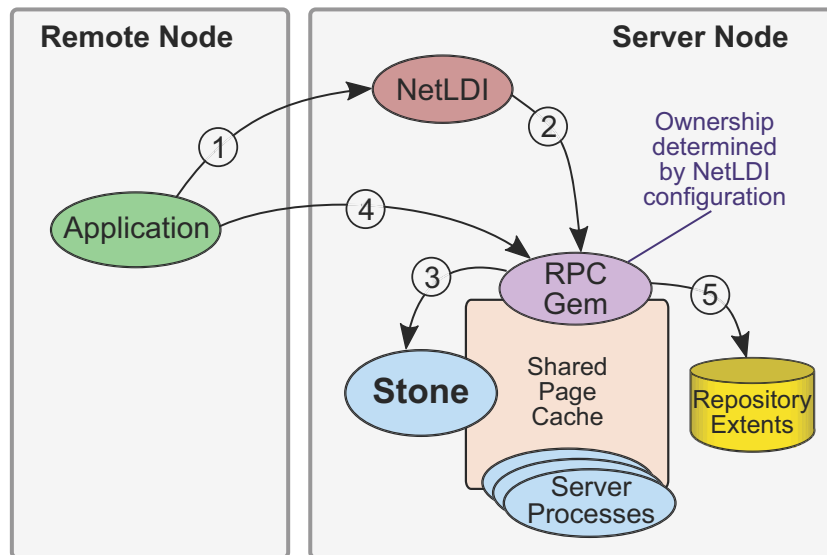
When a remote client applications requests a login, it communicates with the appropriate processes on the server. This process is ordinarily transparent to the user.

The following examples show the sequence of steps that occur during login in order to start the required processes and connect them appropriately.

RPC Application with Gem on Server Node

The simplest configuration for a distributed configuration is a client application running on a remote node, that connects to a RPC Gem session on the server node. The following examples describe the sequence of steps that connect this configuration during login.

Figure 5.9 Connecting an RPC Application with Gem on Server Node



1. The client application contacts the NetLDI on the server node and requests a session.
2. The NetLDI invokes `gemnetobject` or a similar script to start an RPC Gem.

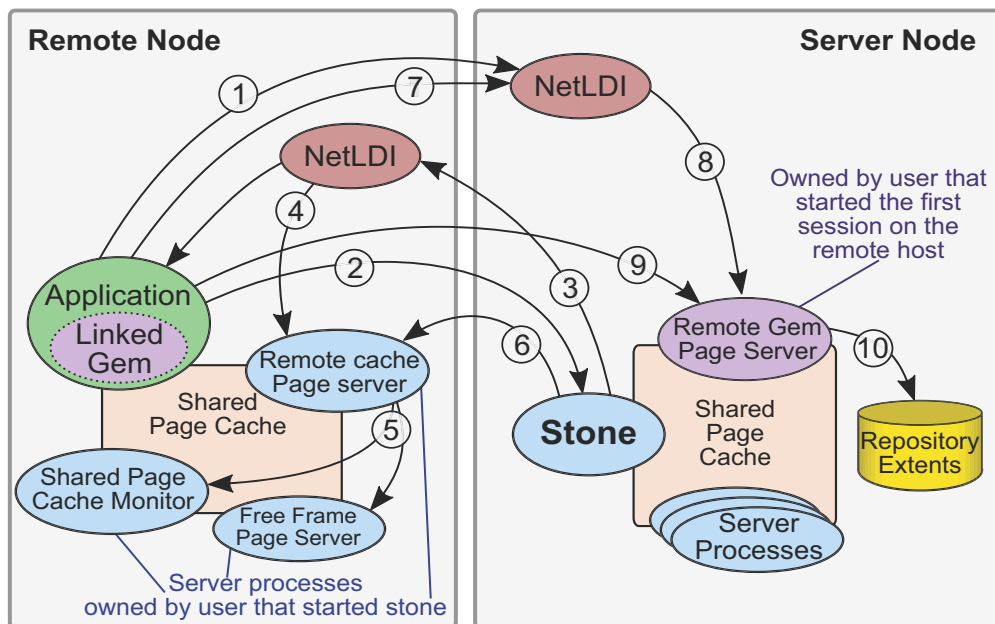
3. The Gem performs the login to the Stone.
4. The application client connects to the Gem.
5. The Gem attaches extent files.

Linked Application on Remote Node

A remote linked login is more involved, since the remote shared page cache and associated processes are also started up.

In this example, there is a NetLDI running on the remote node. This is not required for a linked login; in the case of a linked login with no NetLDI running, the tasks performed by the NetLDI are handled by the linked gem. .

Figure 5.10 Connecting a Linked Application to a Remote Server



1. The client application contacts the NetLDI on the server node, requesting the Stone's listening port.
2. The client application contacts the Stone on its listening port, requesting login.
3. Stone requests the Remote Cache page server on remote node, if there is not already an SPC on the remote node. If there is a cache already there, continue with step 6.
4. The NetLDI on the remote node launches the Remote Cache page server.
5. The Remote Cache page server launches the Shared Page Cache Monitor, which initializes the shared page cache. The Remote Cache pages server also launches the free frame page server.
6. Stone connects to the Remote Cache page server.

7. The client application requests a Remote Gem page server, if there is not already a Remote Gem page server for this node. If a Remote Gem page server already present, continue with step 9.
8. The NetLDI on the server node launches the Remote Gem page server.
9. The client application connects to the Remote Gem page server.
10. A newly launched Remote Gem page server attaches the extent files.

If You Still Have Trouble

If you still can't log in to GemStone from an application on a remote node, try logging in on the server node as the same UNIX user account. We suggest that you first try a linked application, such as **topaz -l**, and when that works, move on to an RPC application (such as **topaz** or the equivalent **topaz -r**), still on the server.

Try Linked Topaz on the Server

A linked application on the server offers the least complicated kind of login because the server's shared page cache is already running and no network facilities are used. Any problems are likely to involve access permission for the shared page cache or the repository extents, which can also block attempts to log in from a client node.

- ▶ Make sure that the owner of the topaz process (`$GEMSTONE/bin/topaz`) can access the shared page cache. Use the UNIX command **ipcs -m** to display permissions, owner, and group for shared memory; for example:

```
Server% ipcs -m
IPC status from <running system> as of Thu Aug 17 16:22:27 PDT
2017
T      ID      KEY          MODE          OWNER        GROUP
Shared Memory:
m      768 0x4c177155  --rw-rw----  gsadmin      pubs
```

Compare the owner and group returned by **ipcs** with the owner of the Topaz process. You can use the **ps** command to determine the owner; for example, **ps -ef | grep topaz**. (The switches may be different on your system.)

A typical problem arises when root owns the Stone process and the shared page cache because their group ordinarily will be a special one to which Topaz users do not belong. Related problems may occur with a linked GemBuilder session. The third-party Smalltalk may be installed without the S bits and therefore may rely on group access to the shared page cache and repository. For background information, see "File Permissions" on page 80.

To correct a shared page cache access failure, either change the owner and group of the setuid files or have the Stone started by a user whose primary group is one to which other GemStone users belong. Unlike file permissions, the shared page cache permissions cannot be set directly.

- ▶ Make sure the owner of the Topaz process has read-write access to `$GEMSTONE/data/extent0.dbf`.

Try Topaz RPC on the Server

The next step should be to try running Topaz on the server with a separate Gem session process. This configuration relies on the NetLDI to start a Gem session process, and that

process, not the application itself, must be able to access the shared page cache and repository extent.

- ▶ Make sure that a NetLDI is running on the server by invoking **gslist**. The default name is `gs64ldi`. If you need to start a NetLDI, the command is **startnetldi**.

GemStone uses the NetLDI to start a Gem session process that does repository I/O in this configuration. For the NetLDI to start processes for anyone other than its owner, it must be owned by root or it must be started in guest mode and captive account mode by someone logged in as the captive account.

The NetLDI writes a log file with the default name `/opt/gemstone/log/gs64ldi.log`; this may be overridden by the **startnetldi -l** option. Using the **gslist -x** command will provide the location of all log files. The log file contents may help you diagnose problems. (See the following discussion, “Check NetLDI Log Files.”)

- ▶ Make sure that the owner of the resulting Gem session process (`$GEMSTONE/sys/gem`) can access the shared page cache and `extent0.dbf` through group membership or S bits. The troubleshooting is the same as that given on page 99 for the `topaz` executable.
- ▶ The user who starts `topaz` (or the NetLDI captive account when it is in use) must have write permission for `$HOME` so that the session process can create a log file there. (For a workaround for situations where write permission is not allowed, see “Controlling log file names and locations” on page 371.)

Check NetLDI Log Files

Troubleshooting on a distributed GemStone system can be complicated. What looks like a hung process may actually be caused by incorrect NRS syntax or by another node on the network going down. The information for analyzing problems may be found in log files on all the nodes used by GemStone.

Where the log file messages include NRS strings, be sure to check their syntax. The problem may be as simple as an incorrect NRS or one that was not expanded by the shell as you intended.

If you can't identify the problem from the standard log messages, try running the NetLDI in debug mode, which puts additional information in the log. The command line is **startnetldi [netLdiName] -d**. For further details, see **startnetldi** on page 353.

5.4 Tuning for Distributed Configurations

A number of configuration options take effect specifically when configurations include remote Gems or caches.

- ▶ With very large caches, the remote page server for a Gem will improve performance by increasing the value of the `GEM_PGSRV_FREE_FRAME_CACHE_SIZE` configuration option. For example:

```
GEM_PGSRV_FREE_FRAME_CACHE_SIZE = 25;
```

- ▶ To improve performance on remote caches, set the value of the `GEM_PGSRV_UPDATE_CACHE_ON_READ` configuration option to True so that remote Gem sessions will update their local caches. For example:

```
GEM_PGSRV_UPDATE_CACHE_ON_READ = TRUE;
```


Running GemStone

This chapter shows you how to perform some common GemStone/S 64 Bit system operations:

Starting the GemStone Server (page 103)

How to startup the GemStone repository, and troubleshooting Stone startup failures.

Starting a NetLDI (page 109)

How to start a NetLDI, and troubleshooting NetLDI startup failures.

Starting a GemStone Session (page 110)

How to login and troubleshooting login failures

Shutting Down Sessions, the Object Server, and NetLDI (page 116)

How to stop sessions and shut down server processes

Recovering from an Unexpected Shutdown (page 117)

Troubleshooting unexpected shutdowns.

6.1 Starting the GemStone Server

In order to start a Stone repository monitor, the following must be identified through your operating system environment:

▶ **Where GemStone is installed**

The GEMSTONE environment variable must point to the directory where GemStone is installed, such as `/users/gemstone`. The directory `$GEMSTONE/bin` should be in your search path for commands.

▶ **Which configuration parameters to use**

The repository monitor must find a configuration file. The default is `$GEMSTONE/data/system.conf`. Other files can supplement or replace the default file; for information, see “How GemStone Uses Configuration Files” on page 285.

▶ **Which repository to use**

The configuration file must give the path to one or more repository files (extents) and to space for transaction logs. The default configuration file specifies `$GEMSTONE/data/extent0.dbf` for the extent file, and places transaction logs in `$GEMSTONE/data/`. You may want to move these files to other locations. For further information, see “Choosing the Extent Location” on page 45.

To Start GemStone

Follow these steps to start GemStone following installation or an orderly shutdown. (To recover from an abnormal shutdown, refer to “Recovering from an Unexpected Shutdown” on page 117.)

Step 1. Set the `GEMSTONE` environment variable to the *full pathname* (starting with a slash) of the directory where GemStone is installed. Ordinarily this directory has a name like `GemStone64Bit3.4.0-x86_64.Linux` (depending on your platform). For example:

```
$ GEMSTONE=/users/GemStone64Bit3.4.0-x86_64.Linux
$ export GEMSTONE
```

If you have been using another version of GemStone, be sure you update or unset previous settings of these environment variables:

- ▶ `GEMSTONE`
- ▶ `GEMSTONE_SYS_CONF`
- ▶ `GEMSTONE_EXE_CONF`
- ▶ `GEMSTONE_NRS_ALL`

Step 2. Set your UNIX path. One way to do this is to use one of the `gemsetup` scripts. There is one version for users of the Bourne and Korn shells and another for users of the C shell. These scripts also set your man page path to include the GemStone man pages. Note that these scripts append to the end of your path or man path; you will need to manually remove references to older versions of GemStone.

```
(Bourne or Korn shell)
$ . $GEMSTONE/bin/gemsetup.sh
or (C shell)
% source $GEMSTONE/bin/gemsetup.csh
```

Step 3. Start GemStone by using the `startstone` command:

```
% startstone [gemStoneName]
```

where *gemStoneName* is optional and is the name you want the repository monitor to have. The default name is `gs64stone`. See `startstone` on page 355 for additional information.

To Troubleshoot Stone Startup Failures

If the Stone repository monitor fails to start in response to a `startstone` command, it’s likely that the cause is one of the following. Inspect the Stone log for clues (the default location is `$GEMSTONE/data/gs64stone.log`).

- ▶ The GemStone key file is missing or invalid.
- ▶ The shared page cache cannot be attached.

- ▶ A problem with the extents: a missing extent or one that is in use by another process.
- ▶ A problem with transaction logs: a log needed for recovery is missing, or the log directory or device does not exist.
- ▶ The repository has become corrupted.

Missing or Invalid Key File

The Stone repository monitor must be able to read the GemStone key file. By default, this is `$GEMSTONE/sys/gemstone.key`. The location and filename can be configured by the `KEYFILE` configuration parameter.

Ordinarily, you create the key file during installation from information provided by GemStone. Be careful to enter the information correctly. GemStone key files are platform-specific, and key files for earlier versions may not work with new major releases.

If you do not have a valid key file, contact GemStone Technical Support as described under “Technical Support” on page 4.

Shared Page Cache Cannot Be Attached

The shared page cache monitor must be able to create and attach to the shared memory segment that will serve as the shared page cache. Several factors may prevent this from happening:

- ▶ On some platforms, shared memory is not enabled in the kernel by default, or its default maximum size is too small to accommodate the GemStone configuration. GemStone’s default configuration requires a shared memory segment somewhat larger than 75 MB. For specifics about configuring shared memory, refer to the *GemStone/S 64 Bit Installation Guide* for your platform.
- ▶ If the size of the shared page cache has been increased, the operating system’s limit on shared memory regions may need to be increased accordingly. GemStone includes a utility, `$GEMSTONE/install/shmem`, that will help you check the configuration; this is described on page 42.
- ▶ The repository executables (the Stone, Gems, and Page servers) must have permission to read and write the shared page cache. Ways to set up access are described in “How To Set Up a Raw Partition” on page 52. In general, users must belong to the same group as the Stone repository monitor. If the Stone is running as root, it is unlikely that other users will be able to access the shared page cache.

Extent Missing or Access Denied

If the Stone repository monitor cannot access a repository extent file, it logs a message like the following:

```
GemStone is unable to open the file
!TCP@pelican#dbf!/users/GemStone/data/extent0.dbf.
reason = File = /users/GemStone/data/extent0.dbf
DBF Op: Open; DBF Record: -1;
Error: open() failure; System Codes: errno=2, ENOENT, The file or
directory specified cannot be found
```

An error occurred opening the repository for exclusive access.

Stone startup has failed.

Examine the message for further clues. The extent file could be missing, the permissions on the file or directory could be set incorrectly, or there may be an error in the configuration file that points to the extents. Correct the problem, then try starting GemStone again.

Extent Open by Another Process

If another process has an extent file open when you attempt to restart GemStone, a message like the following appears in the Stone log (by default, `$GEMSTONE/data/gs64stone.log`):

```
GemStone is unable to open the file
!TCP@pelican#dbf!/users/GemStone/data/extent0.dbf.
reason = File = /users/GemStone/data/extent0.dbf
DBF Op: Open; DBF Record: -1; Error: exclusive open: File is open
by another process.; System Codes:
errno=11, EAGAIN, No more processes (due to process table full,
user quotas, or insufficient memory)
```

An error occurred opening the repository for exclusive access.

Stone startup has failed.

Close any other Gem sessions (including Topaz sessions) that are accessing the repository you are trying to restart, or wait for a **copydbf** to complete. Use **ps -ef** (the options on your system may differ) to identify any `pgsvrmain` processes that are still running, and then use **kill processid** to terminate them. Try again to start GemStone.

Extent Already Exists

If GemStone attempts to recover from a system crash that occurred just after an extent was created, and GemStone was not able to write a checkpoint when the extent was added, you will find an error message like the following in the Stone log:

```
An error occurred in recovery for extentId 2:
fileName= !TCP@pelican#dbf!/users/GemStone/data/extent1.dbf
File already exists; you must delete it before recovery can
succeed.
```

Check that an extent was being added to the repository at or shortly before the crash. If necessary, look for a message near the end of the Stone log file.

- ▶ If an extent was being added, there is no committed data in the extent file yet. Delete the specified file and do not replace it with anything. Try to start GemStone again. The recovery procedure will recreate the extent file.
- ▶ If an extent was NOT being added, it is possible that an existing extent has been corrupted. For instance, `extent0.dbf` of a multiple-extent repository may have been overwritten. Try to determine the cause and whether the action can be rectified. You may have to restore the repository from a backup.

Other Extent Failures

At startup, the GemStone system performs consistency checks on each extent listed in `DBF_EXTENT_NAMES`.

All extents must have been shut down cleanly with a repository checkpoint the last time the system was run. This consistency check is the only one for which GemStone attempts automatic recovery.

The following consistency checks, if failed, cause the startup sequence to terminate. These failures imply corruption of the disk or file system, or that the extents were modified at the operating system level (such as by `cp` or `copydbf`) outside of GemStone's control and in a manner that has corrupted the repository.

- ▶ Extents must be in proper sequence within `DBF_EXTENT_NAMES`.
- ▶ Extents must be properly sequenced in time.
- ▶ The last checkpoint must have occurred earlier than or at the same time as the current system time (in GMT).
- ▶ Extents must belong to the correct repository.

Transaction Log Missing

If GemStone cannot find the transaction log file for the period between the last checkpoint and an unexpected shutdown, it puts a message like this in the Stone log:

```
Extent 0 was not cleanly shutdown; recovery is needed.  
<Repository startup statistics>
```

```
Repository startup is from checkpoint = (fileId 6, blockId 3)
```

```
ERROR: cannot find log file(s) to recover repository.  
To proceed without tranlogs and lose transactions committed  
since the last checkpoint use "-N" switch on your startstone  
command.
```

```
An error occurred when attempting to start repository recovery.  
Waiting for aiowrites to complete
```

```
Stone startup has failed.
```

If the log file was archived and removed from the log directory, restore the file.

If the log file is no longer available, you can use **startstone -N** to restart from the most recent checkpoint in the repository. However, any transactions that occurred during the intervening period cannot be recovered.

NOTE

*When you use **startstone** with the **-N** option, any transactions occurring after the last checkpoint are permanently lost.*

Other Startup Failures

- ▶ Check `/opt/gemstone/locks` (or equivalent location, as discussed on page 31) and remove old files. On Solaris systems, also check `/tmp/gemstone` for `stoneName . . FIFO`.
- ▶ Certain unexpected shutdowns may leave UNIX interprocess communication facilities allocated, which can block attempts to restart the repository monitor. Use the command **ipcs** to identify the shared memory segments and semaphores allocated, then use **ipcrm** to free those resources allocated to a repository monitor that is no longer running. For information about **ipcs** and **ipcrm**, consult your operating system's documentation.
- ▶ If it takes more than 5 minutes for your cache to complete initialization, the startup timeout may be expiring. Set the environment variable `$GEMSTONE_SPCMON_STARTUP_TIMELIMIT` (page 388).
- ▶ Check your installation configuration and make sure that all required files and libraries are present and uncorrupted.
- ▶ Try to run **pageaudit** on the repository. (See "Repository Page and Object Audit" on page 132.)

If you are still unable to start GemStone or determine the reason that startup is failing, contact your local GemStone administrator or GemStone Technical Support.

If this is an existing GemStone repository and the problems reported on startup attempts indicate that the repository is corrupt, you may need to restore from backups, as described in Chapter 11. See "How to Restore from Backup" on page 219.

Listing Running Servers

The **gslist** utility lists all Stone repository monitors, shared page cache monitors, and NetLDIs that are running. The **gslist** command by itself checks the locks directory (`/opt/gemstone/locks`, `/usr/gemstone/locks`, or `$GEMSTONE_GLOBAL_DIR/locks`) for entries. The **-v** option causes it to verify that each process is alive and responding. For example:

```
% gslist -v
Status Version Owner Started Type Name
-----
OK 3.4.0 gsadmin Aug 11 12:02 cache gs64stone~1c9fa07f0412665
OK 3.4.0 gsadmin Aug 11 12:02 Stone gs64stone
OK 3.4.0 gsadmin Aug 11 10:13 Netldi gs64ldi
```

By default, **gslist** lists servers on the local node. The **-m host** option performs the operation on node *host*, which must have a compatible NetLDI running.

6.2 Starting a NetLDI

You will usually need to start a GemStone Network Long Distance Information (NetLDI) server when starting a Stone repository monitor. NetLDI servers are needed to start up Gem processes for RPC logins, and for starting up caches on behalf of Gems that are on other nodes.

If you are running distributed configurations, you will need to perform these steps on each node that requires a NetLDI.

To start a NetLDI server, perform the following steps on the node where the NetLDI is to run:

Step 1. Set the GEMSTONE environment variable to the *full pathname* (starting with a slash) of the directory where GemStone is installed. Ordinarily this directory has a name like `GemStone64Bit3.4.0-x86_64.Linux` (depending on the platform). For example:

```
$ GEMSTONE=/installDir/GemStone64Bit3.4.0-x86_64.Linux
$ export GEMSTONE
```

If you have been using another version of GemStone, be sure you update or unset previous settings of the `$GEMSTONE_NRS_ALL` environment variable

Step 2. Use one of the `gemsetup` scripts to set your UNIX path. There is one version for users of the Bourne and Korn shells and another for the C shell. These scripts also set your man page path to include the GemStone man pages.

(Bourne or Korn shell)

```
$ . $GEMSTONE/bin/gemsetup.sh
```

or (C shell)

```
% source $GEMSTONE/bin/gemsetup.csh
```

Step 3. Start the NetLDI by using the `startnetldi` command.

❑ To start the NetLDI for password authentication, make sure that `$GEMSTONE/sys/netldid` is owned by root and has the S bit set. Issue this command (on some operating systems, you may have to issue it as root):

```
% startnetldi
```

❑ To start the NetLDI in guest mode (authentication is not required), make sure that `$GEMSTONE/sys/netldid` does NOT have the S bit set. Log in as the captive account *name*, then issue this command:

```
% startnetldi -g -aname
```

See `startnetldi` on page 353 for additional command arguments and further detail. For information about the authentication modes, see under “Configuration Decisions” on page 75.

To Troubleshoot NetLDI Startup Failures

If the NetLDI service fails to start in response to a `startnetldi` command, it’s likely that the cause is one of the following:

- ▶ The NetLDI is to run as root but the guest mode option is specified. This combination is not allowed.

- ▶ The account starting the NetLDI does not have permission to create or append to its log file.
- ▶ The account starting the NetLDI does not have read and execute permission for `$GEMSTONE/sys/netldid`.

Check the NetLDI log for clues. By default, the NetLDI log (`netLdiName.log`) is located in `/opt/gemstone/log/`. On some systems, this file may be located in `/usr/gemstone/log/`, and may be overridden using the `startnetldi -l` argument, or by setting `$GEMSTONE_GLOBAL_DIR`.

6.3 Starting a GemStone Session

This section tells how to start a GemStone session and log in to the repository monitor. The instructions apply to all logins from the node on which the Stone repository monitor is running.

- ▶ For additional information about the GemStone administrative logins, see Chapter 8, “User Accounts and Security”.
- ▶ For additional information about logging in from a remote node, see Chapter 5, “Connecting Distributed Systems”.

This section begins with a brief discussion of environmental variables, and then presents two examples. The first example starts a linked application and logs in to GemStone. The second example starts an RPC application, which in turn spawns a separate Gem session process that communicates with the GemStone server.

The examples use Topaz as the application because it is part of the standard GemStone Object Server distribution. Other applications may use different steps to accomplish the same purpose. Some users may prefer to make these steps part of an initialization file.

For an explanation of the difference between linked and RPC sessions, see “Linked and RPC Applications” on page 64.

To Define a GemStone Session Environment

In order to start a GemStone session, the following must be defined through your operating system environment:

- ▶ Where GemStone executables and libraries are installed.

All GemStone users must have a `GEMSTONE` environment variable that points to the GemStone installation directory, such as `/installDir/GemStone64Bit3.4.0-x86_64.Linux` (depending on your platform). The directory `$GEMSTONE/bin` should be in your search path for commands..

- ▶ Which configuration parameters to use.

While system defaults, or a system-wide configuration file, can be used to configure Gem sessions, you may want to configure individualized environments and configuration files for specific sessions. This may involve setting an environmental variable, such as `GEMSTONE_EXE_CONF`. For further information, see “How GemStone Uses Configuration Files” on page 285.

To Start a Linked Session

The following steps show how to start a linked application (here, the linked version of Topaz). The steps for setting the GEMSTONE environment variable and the operating system path for a session are the same as those given on page 104 for starting a repository monitor. They are repeated here for convenience.

The procedure assumes that the Stone repository monitor has already been started and has the default name *gs64stone*.

Step 1. Set the GEMSTONE environment variable to the *full pathname* (starting with a slash) of the directory where GemStone is installed. Ordinarily this directory has a name like *GemStone64Bit3.4.0-x86_64.Linux* (depending on your platform). For example:

```
$ GEMSTONE=/installDir/GemStone64Bit3.4.0-x86_64.Linux
$ export GEMSTONE
```

If you have been using another version on GemStone, be sure you update or delete previous settings of these environment variables:

- ▶ GEMSTONE
- ▶ GEMSTONE_SYS_CONF
- ▶ GEMSTONE_EXE_CONF
- ▶ GEMSTONE_NRS_ALL

Step 2. Set your UNIX path. One way to do this is to use one of the *gemsetup* scripts. There is one version for users of the Bourne and Korn shells and another for users of the C shell. These scripts also set your man page path to include the GemStone man pages.

(Bourne or Korn shell)

```
$ . $GEMSTONE/bin/gemsetup.sh
```

or (C shell)

```
% source $GEMSTONE/bin/gemsetup.csh
```

Step 3. Start linked Topaz:

```
% topaz -l
```

Step 4. Set the *UserName* login parameter:

```
topaz> set username DataCurator
```

Step 5. Log in to the Gem session. It will query you for the password.

```
topaz> login
GemStone Password?
[Info]: LNK client/gem GCI levels = 34001/34001
--- 09/20/17 17:16:35 PDT Login
[Info]: Loaded /benton2/users/lalmarod/GS6434/lib/libfloss-
3.4.0-64.so
[Info]: User ID: DataCurator
[Info]: Repository: stone34
[Info]: Session ID: 5
[Info]: GCI Client Host: <Linked>
[Info]: Page server PID: -1
[Info]: using libicu version 58.2
[Info]: Gave this process preference for OOM killer: wrote to
/proc/26091/oom_score_adj value 250
[09/20/17 17:16:35 PDT ]
  gci login: currSession 1  linked session
successful login
topaz 1>
```

At this point, you are logged in to a Gem session process, which is linked with the application. The session process acts as a server to Topaz and as a client to the Stone. Information about Topaz is in the manual *GemStone Topaz Programming Environment*.

When you are ready to end the GemStone session, you can log out of GemStone and exit Topaz in one step by invoking the Topaz **exit** command:

```
topaz 1> exit
```

To Start an RPC Session

The following steps show how to start an RPC application (here, the RPC version of Topaz) on the server node. The procedure assumes that the Stone is running under the default name *gs64stone* and that you are already set up to run a GemStone session as described in Step 1 and Step 2 of the previous example (“To Start a Linked Session”).

Sessions that login RPC use SRP (Secure Remote Password) and SSL to authenticate passwords for login. If the Gem is running on the server node, the connection reverts to normal socket communication after login completes.

The following steps demonstrate an RPC login from topaz:

Step 1. Use **gslist** to find out if a NetLDI is already running. The default name for the NetLDI is *gs64ldi*.

```
% gslist
Status Version  Owner      Started   Type  Name
-----
exists 3.4.0   gsadmin  Aug 11 12:02  cache gs64stone~1c9fa07f0412665
exists 3.4.0   gsadmin  Aug 11 12:02  Stone  gs64stone
exists 3.4.0   gsadmin  Aug 11 10:13  Netldi gs64ldi
```

If necessary, start a NetLDI following the instructions under “Starting a NetLDI” on page 109.

Step 2. Start the RPC application (such as Topaz), then set the *UserName*.

```
topaz> set username DataCurator
```

Step 3. Unless the NetLDI is running in guest mode with a captive account, set the application login parameters, such as *HostUserName* and *HostPassword*, after you start the application. For example:

```
topaz> set hostusername yourUnixId
topaz> set hostpassword yourPassword
```

Step 4. Set *GemNetId* (the name of the Gem service to be started) to *gemnetobject*. This script starts the separate Gem session process for you. For example:

```
topaz> set gemnetid gemnetobject
```

Step 5. Log in to the GemStone session.

```
topaz> login
GemStone Password?
[Info]: Loaded /benton2/users/lalmarod/GS6434/lib/libfloss-
3.4.0-64.so
[08/12/2017 11:36:47.777 PDT]
  gci login: currSession 1 rpc gem processId 6943 socket 6
successful login
topaz 1>
```

At this point, you are logged in through a separate Gem session process that acts as a server to Topaz RPC and as a client to the Stone repository monitor.

When you are ready to end the GemStone session, you can log out of GemStone and exit Topaz by in one step by invoking the Topaz **exit** command:

```
topaz 1> exit
```

To Troubleshoot Session Login Failures

Several factors may prevent successful login to the repository:

- ▶ Your GemStone key file may establish a maximum number of user sessions that can simultaneously be logged in to GemStone. (Note that a single user may have multiple GemStone sessions running simultaneously.) The limit itself is encoded in the keyfile used to start the stone (by default, `$GEMSTONE/sys/gemstone.key`), and reported in the stone log on startup. Look for a line like this:

```
SESSION MAX: The licensed concurrent session max is 10.
```

- ▶ The `STN_MAX_SESSIONS` configuration option can restrict the number of logins to fewer than a particular key file allows. An entry in the Stone log file shows the maximum at the time the Stone started. Look for a line like this:

```
SESSION CONFIGURATION: The maximum number of concurrent
sessions is 40
```

- ▶ The `SHR_PAGE_CACHE_NUM_PROCS` configuration option restricts the number of sessions that can attach to a particular shared page cache. This is normally computed based on the setting for `STN_MAX_SESSIONS`.

Multi-threaded operations use additional slots for their working threads while they are executing. If you are close to your session limit, these operations may prevent other sessions from logging in.

- ▶ The UNIX kernel must provide sufficient semaphores and file descriptors for each logged in session. See your *Installation Guide* for information on UNIX kernel tuning that may be necessary.
- ▶ The owner of the Gem or a linked application process must have write access to the extent file and to the shared page cache. Use the UNIX command **ipcs -m** to display permissions, owner, and group for shared memory. For example:

```
server% ipcs -m
IPC status from <running system> as of Mon August 9 16:21:08
PDT 2017
T      ID      KEY          MODE          OWNER        GROUP
Shared Memory:
m      25089    0x4c000ed5  --rw-rw----  gsadmin      users
```

Typical problems occur with linked applications, which may be installed without the S bit and therefore rely on group access to the shared page cache and the repository.

- ▶ If the session is using a separate (RPC) gem process, see “Troubleshooting Remote Logins” on page 96.

Identifying and Stopping Logged-in Sessions

Privileges required: SessionAccess.

To identify the sessions currently logged in to GemStone, send the message `System class>>currentSessionNames`. This message returns an array of internal session numbers and the corresponding `UserId`. For example:

```
topaz 1> printit
System currentSessionNames
%
session number: 2    UserId: GcUser
session number: 3    UserId: GcUser
session number: 4    UserId: SymbolUser
session number: 5    UserId: DataCurator
```

The session number can be used with other `System` class methods to stop a particular session or to obtain its `UserProfile`.

To get the `sessionId` for the current executing session, use `System class >> session`. For example,

```
topaz 1> printit
System session
%
5
```

To get the `UserProfile` for a given session, execute:

```
System userProfileForSession:aSessionId
```

To get the `UserProfile` for the current session, execute:

```
System myUserProfile
```

The method `System class>>descriptionOfSession:aSessionId` returns an array of descriptive information, which can be used to find out details information and status for

any session. This method returns an Array of 21 elements; the values in each slot are defined as follows:

1. The UserProfile of the session; nil if the UserProfile is recently created and not visible from this session's transactional view or the session is in login or processing, or has logged out.
2. A SmallInteger, the process ID of the Gem or topaz -l process .
3. The hostname of the machine running the Gem process. Specifically, the peer's hostname as seen by stone, for the gem to stone network connection used for login. (a String, limited to 127 bytes).
4. Primitive number in which the Gem is executing, or 0 if it is not executing in a long primitive.
5. Time of the session's most recent beginTransaction, commitTransaction, or abortTransaction (from System timeGmt).
6. The session state (a SmallInteger).
7. A SmallInteger whose value is -1 if the session is in transactionless mode, 0 if it is not in a transaction and 1 if it is in a transaction.
8. A Boolean whose value is true if the session is currently referencing the oldest commit record, and false if it is not.
9. The session's serial number (a SmallInteger).
10. The session's sessionId (a SmallInteger).
11. A String containing the IP address of host running the GCI process. If the GCI application is remote, the peer address as seen by the gem of the GCI application to gem network connection. If the GCI application is linked (using libgcilnk*.so or gcilnk*.dll) this is the peer's IP address as seen by stone, for the gem to stone network connection used for login.
12. The priority of the session (a SmallInteger).
13. Unique host ID of the host where the session is running (an Integer)
14. Time of the session's most recent request to stone (from System timeGmt)
15. Time the session logged in (from System timeGmt)
16. Number of commits which have occurred since the session obtained its view.
17. Nil or a String describing a system or gc gem .
18. Number of temporary (uncommitted) object IDs allocated to the session.
19. Number of temporary (non-persistent) page IDs allocated to the session.
20. A SmallInteger, 0 session has not voted, 1 session voting in progress, 2 session has voted, or voting not active.
21. A SmallInteger, processId of the remote GCI client process, or -1 if the session has no remote GCI client.
22. The KerberosPrincipal object used for passwordless login to the session, or nil if passwordless login was not used.

Refer the image method comment for the most recent details.

6.4 Shutting Down Sessions, the Object Server, and NetLDI

Stopping Logged-in Sessions

Privileges required: SessionAccess and SystemControl

There are a number of methods on System class that can be used to stop a specific session, or all sessions:

`stopSession: aSessionId`

Stop the specified session; any transactions that the session was in are aborted, and the session is terminated. This method does not stop the GcGems or SymbolGem.

`terminateSession: aSessionId timeout: timeoutSeconds`

Stop the specified session; any transactions that the session was in are aborted, and the session is terminated. Waiting up to *timeoutSeconds* for the session to complete terminating before returning. This method can be used to stop the GcGems. but not the SymbolGem.

`stopUserSessions`

Stops all sessions other than system Gems; does not stop the GcGems nor SymbolGem. Any transactions that any of the sessions were in are aborted.

NOTE

Be aware that it may take as long as a minute for a session to terminate after you send `stopSession`. If the Gem is responsive, it usually terminates within milliseconds. However, if a Gem is not active (for example, sleeping or waiting on I/O), the Stone waits one minute for it to respond before forcibly logging it out. You can bypass this timeout by sending `terminateSession:timeout:`

To verify all user sessions have logged out or been terminated, send the message `currentSessionNames` to System. For example, using Topaz:

```
topaz 1> printit
System currentSessionNames
%
session number: 2      UserId: GcUser
session number: 3      UserId: GcUser
session number: 4      UserId: SymbolUser
session number: 5      UserId: DataCurator
```

The SymbolUser and GcUser sessions are system session and will be shut down cleanly when the stone is shut down. The above example includes session 5, which is the user executing the example code.

Stopping the Stone

After all user sessions have logged out, use the **stopstone** command, which performs an orderly shutdown in which all committed transactions are written to the extent files.

```
% stopstone [gemStoneName] [-i]
```

If you do not supply the name of the Stone repository monitor, **stopstone** prompts you for one.

The **-i** option aborts all current (uncommitted) transactions and terminates all active user sessions. If you do not specify this option and other sessions are logged in, GemStone will not shut down and you will receive a message to that effect.

stopstone prompts you to supply a GemStone username and password. The user must have the SystemControl privilege (initially, this privilege is granted to SystemUser and DataCurator).

Stopping the NetLDI

There is a similar command to shut down the NetLDI network service.

```
% stopnetldi [netLdiName]
```

For more information, see the command reference in Appendix B; **stopstone** on page 363 and **stopnetldi** on page 362.

If you are logged in to a GemStone session, you can invoke `System class>>shutDown`, which also requires the SystemControl privilege.

Using OS kill

If you must halt a specific Gem session process or GemStone server processes, be sure to use only **kill** or **kill -term** so that the Gem or other process can perform an orderly shutdown.

kill -usr1 will not kill the process, but will cause a GemStone process to write its C and Smalltalk call stacks to the process log file. For linked logins, which do not have a separate process, the stack is written to the application's stdout.

Do NOT use **kill -9** or another uncatchable signal, which does not result in a clean shutdown, unless it is unavoidable. On some platforms, particular failures in disk I/O can result in a process that does not respond to kill.

If for some reason you do need to send **kill -9** to a shared page cache monitor, use **ipcs** and **ipcrm** to identify and free the shared memory and semaphore resources for that cache. If you send kill -9 to a Stone, use **ipcs** to determine whether **ipcrm** should be invoked.

6.5 Recovering from an Unexpected Shutdown

GemStone is designed to shut down in response to certain error conditions as a way of minimizing damage to the repository. If GemStone stops unexpectedly, it probably means that one of the following situations has occurred:

- ▶ Disk failure
- ▶ Shared page cache monitor failure
- ▶ Fatal error detected by a Gem
- ▶ File system corruption
- ▶ Power failure
- ▶ Operating system crash

When GemStone shuts down unexpectedly, check the message at the end of the Stone log file to begin diagnosing the problem. By default, the Stone log is

`$(GEMSTONE)/data/gemStoneName.log`, but there are a number of ways that this can be configured. The names and locations of the Stone and other process log files is described under “GemStone Process Logs” on page 122.

Once the problem is identified, your recovery strategy should take into account the interdependence of GemStone system components. For instance, if an extent becomes unavailable, to restart the system and recover you may have to kill the Stone repository monitor if it is still running. The **stopstone** command won’t work in this situation, since the orderly shutdown process requires the Stone to clean up the repository before it stops.

Clean Shutdown Message

If you see a shutdown message in the system log file, GemStone has stopped in response to a **stopstone** command or a Smalltalk System shutdown method, or in response to a kill -TERM:

```

--- 09/06/17 11:27:59.977 PDT ---
    Starting checkpoint for clean shutdown.
    Waiting for all tranlog writes to complete before shutdown.
    <other shutdown messages>
    Waiting for NetWrite thread to stop
    Waiting for Page Manager thread to stop

--- 09/06/17 11:28:02 PDT ---
    Now stopping GemStone.
```

After a clean shutdown, restart GemStone in the usual manner. For instructions, see “Starting the GemStone Server” on page 103 of this chapter.

Disk Failure or File System Corruption

GemStone prints several different disk read error messages to the GemStone log file. For example:

```

Repository Read failure,
fileName = !#dbf!/users/gemstone/data/extent0.dbf
PageId = 94
File = /users/gs64stone/data/extent0.dbf
too few bytes returned from read()
DBF Operation Read; DBF record 94, UNIX codes: errno=34,...
    "A read error occurred when accessing the repository."
```

If you see a message similar to the above, or if your system administrator identifies a disk failure or a corrupted file system, try to copy your extents to another node or back them up immediately. The copies may be bad, but it is worth doing, just in case. If you’re lucky, you may be able to copy them back after the underlying problem is solved and start again with the current committed state of your repository.

Otherwise, you may need to restore the repository. For details, see the restore procedures in Chapter 11.

Shared Page Cache Error

If you find a message similar to the following in the GemStone log, the shared page cache (SPC) monitor process (`shrpcmonitor`) died. The SPC monitor log, `$GEMSTONE/data/gemStoneName_pcmomnnnn.log`, may indicate the reason.

```
--- 09/06/17 15:07:19 PDT ---
```

```
The stone's connection to the local shared cache monitor was lost.
```

```
Error Text: 'Network partner has disconnected.'
```

The unexpected shutdown of a Gem process may, in rare cases, result in a “stuck spin lock” error that brings down the shared page cache monitor and the Stone. GemStone uses spin locks to coordinate access to critical structures within the cache. In most cases, the monitor can recover if a Gem dies while holding a spin lock, but not all spin locks can be recovered safely. Stuck spin locks may result from a Gem crash, but a typical cause is the use of **kill -9** to kill an unwanted Gem process. If you must halt a Gem process, be sure to use only **kill** or **kill -TERM** so that the Gem can perform an orderly shutdown.

Use **startstone** to restart GemStone. For instructions, see “Starting the GemStone Server” on page 103.

Fatal Error Detected by a Gem

If a Gem session process detects a fatal error that would cause it to halt and dump a core image, the Stone repository monitor may do the same when it is notified of the event. This response on the part of the Stone is configurable through the `STN_HALT_ON_FATAL_ERR` configuration option. When that option is set to True and a Gem encounters a fatal error, the Stone prints a message like this in its log file:

```
Fatal Internal Error condition in Gem
```

```
when halt on fatal error was specified in the config file
```

By default, `STN_HALT_ON_FATAL_ERR` is set to False. That setting causes the Stone to attempt to keep running if a Gem encounters a fatal error; it is the recommended setting for GemStone in a production system. You can set `STN_HALT_ON_FATAL_ERR` to True during development and testing to provide additional checks for potential risks.

Some Other Shutdown Message

In the event of other shutdown messages in the GemStone log:

1. Consider whether the shutdown might have been caused by a disk failure or a corrupt file system, especially if you see an unexpected message such as `Object not found`. If you suspect one of these conditions, start with a page audit of the repository file (see “Repository Page and Object Audit” on page 132).

If the page audit fails, refer to “Disk Failure or File System Corruption” on page 118, and consult your operating system administrator.

If the audit succeeds, continue to the next step.

2. If you don't suspect disk failure or a corrupt file system, try using **startstone** to restart GemStone. For instructions, see “Starting the GemStone Server” on page 103.
3. If the restart fails, you may have to restore the repository. For details, see the restore procedures in Chapter 11.

No Shutdown Message

If the GemStone log doesn't contain a shutdown message, there has probably been a power failure or an operating system crash. In that event, the Stone repository monitor automatically recovers committed transactions the next time it starts. Use **startstone** to restart GemStone, as described under "Starting the GemStone Server" on page 103. See **startstone** on page 355 for more information on this command.

Monitoring GemStone

A properly configured GemStone repository will run normally with little attention. It is still important to monitor the repository, to catch unexpected problems before they become serious. If you have unexpected problems you will need to examine logs, monitor you system, and perform other analysis. The relevant logs and tools are described in this chapter.

GemStone Process Logs (page 122)

details what logs are created by GemStone/S 64 Bit processes, where they are located, and what configuration is possible.

Repository Page and Object Audit (page 132)

provides instructions on how to perform a page audit and object audit of the repository.

Profiling Repository Contents (page 136)

describes how to analyze the repository contents

Monitoring Performance (page 138)

describes how to monitor the performance of the GemStone server and its clients using GemStone Smalltalk methods.

If you decide to keep a GemStone session running for occasional use, be careful not to leave it in an active transaction. A prolonged transaction can cause an excessive commit record backlog and undesirable repository growth, until you either commit or abort.

NOTE

Monitoring on active repositories should be done in manual transaction mode or transactionless mode. See "Disk Space and Commit Record Backlogs" on page 194.

7.1 GemStone Process Logs

All GemStone processes create log files, including startup configuration information, tracking details on certain operations, and details for any errors that were encountered.

For some kinds of processes, these log files are only interesting if an error occurs, and these log files are deleted when the process exits (provided no error occurs). For other kinds of processes, information reported in the log files may provide diagnostic information for problems that occur later to other parts of the system. The log files for these kinds of processes are not deleted when the process exits.

The log file names and directory locations, and the log file deletion policies, are all configurable if you would prefer to set up a customized way to manage your log files. What is important is to know where your log files are and monitor these logs for error conditions, and to know how to find the relevant logs if a problem occurs.

GemStone log contents and names are UTF-8 encoded.

Finding log files

By default, GemStone writes log files to a number of specific locations:

- ▶ the Stone and associated server process logs are in the `$GEMSTONE/data` directory. The Stone log file name and location can be configured as described in the next section; the logs for the AIO Page Server, Free Frame Page Server, SymbolGem, Page Manager, and Admin and Reclaim Gems are in the same location as their Stone's log.
- ▶ RPC Gem logs, and log files for processes that are running on nodes remote from the Stone are in the `$HOME` directory of the UNIX user. The locations and names can be configured by the NRS used to start the Gem. This is described starting on page 127.
- ▶ The NetLDI log is in `/opt/gemstone/log`. The name and can be configured as described on page 126.

If the given GemStone process is running, you can use the `gslis` (page 341) utility to locate its logs. `gslis -x` displays the location of the current log file for Stones, NetLDIs, logsenders, logreceivers, and the shared page cache monitor.

A number of process log files are deleted automatically when the process exits cleanly, to avoid an excessive number of unimportant log files. Details on specific processes describe the applicable log file deletion policy. These policies can be overridden per process (in most cases). You can force or disable delete using the environment variables `GS_FORCE_CLEAN_LOG_FILE_DELETE` and `GS_KEEP_ALL_LOGS`. Log files for processes that exited with an error are never deleted, and the NetLDI and Stone logs are never automatically deleted.

Stone Log

The log for the Stone repository monitor is always appended to, and is therefore cumulative across runs by default. This log is the first one you should check when a GemStone system problem is suspected. In addition to possible warnings and error messages, the log records the following useful information:

- ▶ The GemStone version.
- ▶ The configuration files that were read at startup and, if the `DUMP_OPTIONS` configuration option is set to True, the resulting Stone configuration.

- ▶ Each startup and shutdown of the Stone, the reason for the shutdown, and whether recovery from transaction logs was necessary at startup.
- ▶ Each expansion of a repository extent and its current size.
- ▶ Each opening of a new transaction log.
- ▶ Each startup and shutdown of each GcGem session, and the corresponding processId.
- ▶ Each #abortErrLostOtRoot sent to a Gem.
- ▶ Each suspension and resumption of logins.
- ▶ Certain changes to the login security system.
- ▶ Each time a backup is started and when the backup is completed.

Log name and location

The Stone log by default is *stonename.log*, where *stonename* is the name of the running Stone repository monitor. If a specific name was not specified for *startstone*, the *stonename* defaults to *gs64stone*.

The Stone log file name and location are determined in the following precedence:

1. A path and filename supplied by **startstone -l logFile**. *logfile* may be a filename, or a relative or absolute path and filename, to which the account starting the Stone has write permission. If *logFile* is a filename only, or not an absolute path, *logFile* is created in the current directory or relative to the current directory.
2. A path and filename specified by the GEMSTONE_LOG environment variable. As with **startstone -l**'s argument, this may be set to a filename or to a relative or absolute path and filename.
3. `$GEMSTONE/data/stonename.log`.

Log file deletion policy

The Stone log is never deleted; each restart appends to an existing log file of the same name, if one exists. The information in this log file may be useful for problem diagnosis for a significant time. If this file becomes too large, or the log file name or location is changed, we recommend archiving the older Stone logs.

Shared Page Cache Monitor Log

The shared page cache monitor log includes, among other things:

- ▶ Its configuration (for remote nodes, this may be different from the configuration on the Stone's node).
- ▶ The number of processes that can attach (which can limit the number of logins).
- ▶ The UNIX identifiers for the memory region and the semaphore array (these identifiers are helpful in the event you must remove them manually using the **ipcrm** command).

Log name and location

The log for the shared page cache monitor on the Stone's machine is located in the same directory as the Stone's log. This log file has a name of the form

stoneName_PIDpcmon.log

Check this log if other messages refer to a shared page cache failure.

When a session logs in from another node, a log is created for the shared page cache monitor on the remote node. This log is located by default in the home directory of the account that started the Stone, but this location can be modified by environment variable settings. The default name is of the form

```
startshrpcmonPIDNode.log
```

where *PID* is the process Id of the monitor process, and *Node* is the name of the remote node.

Log file deletion policy

The shared page cache monitor log is not deleted on exit. A new log is created each time the stone is restarted, and old log files should be manually deleted from time to time.

Admin Gem Log

This log shows the startup value of the Admin Gem parameters that are stored in GcUser's UserGlobals, and any changes to them, and records other Admin Gem functions.

Log name and location

Each time the Stone repository monitor starts an administrative garbage collection session (Admin Gem) process, a new log is created. By default, this log is in the same location as the Stone's log. The location of this log file can be set specifically using the environment variable `$GEMSTONE_ADMIN_GC_LOG_DIR`.

The log name is formed using the pattern:

```
stoneName_PIDadmincgem.log
```

where *stoneName* is the name of the Stone, and *PID* is the process Id of the Admin Gem process.

Log file deletion policy

By default, the AdminGem log is not deleted on clean exit.

The Admin Gem is started using the script `$GEMSTONE/sys/runadmingem`. You may create a customized version of this script, commenting out the line that sets `$GEMSTONE_KEEP_LOG`, to allow this log to be automatically deleted.

Reclaim Gem Log

This log shows the startup value of the Reclaim Gem parameters that are stored in GcUser's UserGlobals, and any changes to them, and records other Reclaim Gem functions.

Log name and location

Each time the Stone repository monitor starts a reclaim garbage collection session (Reclaim Gem) process, a new log is created. By default, this log is in the same location as the Stone's log. The location of this log file can be set specifically using the environment variable `$GEMSTONE_RECLAIM_GC_LOG_DIR`. in the same location as the Stone's log.

The log name is formed using the pattern:

```
stoneName_PIDreclaimcgem.log
```

where *stoneName* is the name of the stone, and *PID* is the process Id of the Reclaim Gem process.

Log file deletion policy

By default, the ReclaimGem log is not deleted on clean exit.

The Admin Gem is started using the script `$GEMSTONE/sys/runreclaimgem`. You may create a customized version of this script, commenting out the line that sets `$GEMSTONE_KEEP_LOG`, to allow this log to be automatically deleted.

Free Frame Page Server Log

This log is not usually of interest, unless errors occur or tuning is required.

Log name and location

The free frame page server is started up on repository startup. Its log file is located in the same directory as the log for the shared page cache monitor. This log file has a name of the form:

```
stoneName_PIDpgsvrff.log
```

where *stoneName* is the name of the stone, and *PID* is the process Id of the free frame page server process.

Free Frame page servers for remote shared caches are located in the `$HOME` directory of the UNIX user, with the name pattern as above.

Log file deletion policy

This log is deleted by default on clean exit.

The Free Frame Page Server is started using the script `$GEMSTONE/sys/runffpgsvr`. You may create a customized version of this script, uncommenting the line that sets `$GEMSTONE_KEEP_LOG`, to allow this log to be retained.

AIO Page Server Log

This log is not usually of interest, unless errors occur or tuning is required.

Log name and location

The AIO page server is started up on repository startup. The log file is in the same directory as the log for the shared page cache monitor. This log file has a name of the form:

```
stoneName_PIDpgsvraio.log
```

where *stoneName* is the name of the stone, and *PID* is the process Id of the AIO page server process.

Log file deletion policy

This log is deleted by default on clean exit.

The Free Frame Page Server is started using the script `$GEMSTONE/sys/runaiopgsvr`. You may create a customized version of this script, uncommenting the line that sets `$GEMSTONE_KEEP_LOG`, to allow this log to be retained.

Page Manager Log

This log is not usually of interest, unless errors occur or tuning is required.

Log name and location

The Page Manager is a thread in the Stone, and is not a separate process, but it writes to a separate log for ease of maintenance. The Page Manager log is located in the same directory as the log for the Stone. This log file has a name of the form:

```
stoneName_PIDpagemanager.log
```

where *stoneName* is the name of the stone, and *PID* is the process Id of the Stone process.

Log file deletion policy

This log is not deleted by default on clean exit. Since it is a thread in the stone, it is not started by a specific script, and will only be deleted on clean exit when `$GS_FORCE_CLEAN_LOG_FILE_DELETE` is set.

Symbol Gem Log

This log is not usually of interest, unless errors occur or tuning is required.

Log name and location

The Symbol Gem log is located in the same directory as the Stone's log, by default. The location of this log file can be set specifically using the environment variable `GEMSTONE_SYMBOL_GEM_LOG_DIR`.

The Symbol Gem log file has a name of the form:

```
stoneName_PIDsymbolgem.log
```

where *stoneName* is the name of the stone, and *PID* is the process Id of the Symbol Gem process.

Log file deletion policy

This log is deleted by default, if the SymbolGem exits cleanly and no (nonfatal) errors were reported during the lifetime of the SymbolGem, and no Symbol GC operations were performed while the SymbolGem was running.

NetLDI Log

By default, the NetLDI log contains only configuration information and error messages. The configuration information reflects the environment at the time the NetLDI was started and the effect of any authentication switches specified as part of the `startnetldi` command.

In some cases it is helpful to log additional information by starting the NetLDI in debug mode (`startnetldi -d`). In this mode, the NetLDI writes a record of each communication to or from all clients to its log. Because the log for NetLDI running in debug mode is much larger, you probably won't want to use this mode routinely.

Log name and location

The NetLDI writes a log file (*netLdiName*.log) in `/opt/gemstone/log` (or an equivalent, as described on page 31) on the node on which it runs.

The `startnetldi` script allows you to specify a log file name and location using the `-l` option, and optionally the name `netLdiName`. If no log file name is specified using the `-l` argument, the default is `/opt/gemstone/log/netLDIName.log`.

Log file deletion policy

The NetLDI log file with the specified or default name is appended to, and is never deleted. You should manually remove outdated messages occasionally.

Gem Logs and logs related to Gem Sessions

The log file written by the Gem includes the Gem's startup configuration details, configuration parameters settings, and login information, as well as the messages generated if an error occurs. This information is important when diagnosing client-related problems.

When the RPC or Linked Gem is not running on the same node as the Stone, the login to GemStone also requires other supporting processes to be spawned. Each of these processes has their own log file.

Linked Gems

Linked logins, in which the Gem is part of the client process, do not write a separate log file to disk. The log file output is sent to stdout of the linked process; for example, the linked topaz console. Topaz command such as **output push** allow this information to be written to disk. See the *Topaz User's Guide* for more information.

RPC Gems on Stone's host

An RPC login spawns a separate Gem session process. When this process is on the same node as the Stone, the RPC Gem can connect directly to the server processes, and does not require further supporting processes to be spawned.

By default, the log file for an RPC Gem is located in the home directory of the account that owns the Gem process, which depends in turn on the NetLDI configuration.

You can change the default location for Gem log files by setting `#dir` or `#log` in the `GEMSTONE_NRS_ALL` environment variable for the NetLDI itself or for individual clients; see "Controlling log file names and locations" on page 371. Alternatively, when you log in to GemStone, you can specify a different network resource string (NRS) in your login parameters.

Remote RPC or Linked Gems

For RPC logins where the Gem is not on the same node as the Stone, or for linked logins that are not on the same node as the Stone, the following additional processes are also spawned:

- ▶ A page server on the server node (if one does not exist already for the remote node). This allows the session to access a repository extent on the server node.
- ▶ A page server on the remote node (if one does not already exist for a previous login), to allow the Stone to start or access a shared page cache on the remote client node.
- ▶ A shared page cache monitor on the remote node, to manage the remote cache on the client's node.

- ▶ A free frame page server for the remote shared page cache.

The default location for the log files of these processes is based on any settings for **#dir** or **#log** that is specified for the Gem, or in the home directory of the account that owns the corresponding process. For the page server on the server node, that account ordinarily is the application user. For the shared page cache monitor and page server on the client node, that account is the one that invoked **startstone**.

The following table shows typical log names for processes related to remote logins, given a Stone named **gs64stone** repository on *node1* with a login from a Gem session process on *node2*.

Typical Name	GemStone Process
gemnetobject27853node2.log	Gem session process on <i>node2</i> (serves an RPC session)
gs64stone_27819cachepgsvr_node2.log	Page server on <i>node2</i> that the repository monitor uses to create and access its shared page cache on <i>node2</i>
gs64stone_27820pcmon_node2.log	Shared page cache monitor on <i>node2</i>
gs64stone_23593pgsvrff_node2.log	Free Frame page server for remote cache on <i>node2</i>
runpgsvrmain12397node1.log	Page server on <i>node1</i> that the Gem session process uses to access the repository extents on <i>node1</i>

Further control over log file location and name

NRS includes the **dir:** and **log:** directives, which allow you to specify the log file name and directory location, including pattern substitution.

The options are described under “Controlling log file names and locations” on page 371.

Logsender and logreceiver Logs

The logsender and logreceiver processes are started only if you are setting up a hot standby system.

Log name and location

Each logsender and logreceiver creates a log file in `/opt/gemstone/log` on the node on which it runs.

The log file’s name, by default, is `logsender_listeningPort.log` or `logreceiver_listeningPort.log`.

This location and name can be overridden by including the option `-llogname` when starting the logsender or logreceiver.

Log file deletion policy

The logsender and logreceiver log files with the specified or default names are appended to, and are never deleted. You should occasionally manually examine these logs and remove outdated messages.

Other Log Files

Other GemStone processes also create log files, which are only of interest if an error occurs; these logs are deleted by default, and you may only ever see them if you use the `$GS_KEEP_ALL_LOGS` environment variable. Others are specific to particular utilities, and details are described in separate parts of this manual.

- ❑ Extent `pregrow` produces log files named `stonename_pidpgsvrPreGrow.log` in the Stone's log file directory.
- ❑ `pageaudit` produces a log file for the audit gem, as well as the log file with audit results.
- ❑ `statmonitor`, when started from the configuration option, invokes the script `runstatmonitor` and creates a log file named `stonename_pidrunstatmonitor_type.log` in the Stone's log file directory, or for remote caches, in the home directory of the corresponding UNIX process owner.
- ❑ cache warmers, from the configuration file option or using `startcachewarmer`, produce log files.

Summary of GemStone Process Log Behaviors

This table provides a summary of the various GemStone process log behaviors.

Table 7.1 GemStone process types and log file details

Process Name	Name and location of log file	Script
Stone	<code>stonename.log</code> ; the default for <code>stonename</code> is <code>gs64stone</code> . Override with <code>-l</code> argument to <code>startstone</code> or <code>\$GEMSTONE_LOG</code> . Log file is never deleted; the same file is appended to with each restart.	<code>startstone</code>
NetLDI	<code>/opt/gemstone/netldiname.log</code> ; the default for <code>netldiname</code> is <code>gs64ldi</code> . Override with <code>-l</code> argument to <code>startnetldi</code> . Log file is never deleted. The same file is appended to with each restart.	<code>startnetldi</code>
Shared Page Cache Monitor	<code>stonename_pidpcmon.log</code> , in Stone's log file directory. Remote SPC logs are <code>\$HOME/stonename_pidpcmon_node.log</code> . Not deleted by default.	<code>startshrpcmon</code>
Page Manager (Thread in Stone)	<code>stonename_pidpagemanager.log</code> , in Stone's log file directory. Not deleted by default.	(thread in stone)
AIO page server	<code>stonename_pidpgsvraio.log</code> in Stone's log file directory. Deleted by default on clean exit	<code>runaiopgsvr</code>

Table 7.1 GemStone process types and log file details

Free frame page server	<i>stonename_pidpgsvrff.log</i> . Deleted by default on clean exit. Remote free frame page server logs are <i>\$HOME/stonename_pidpgsvrff_node.log</i> .	runffpgsvr
Symbol Gem	<i>stonename_pidsymbolgem.log</i> in Stone's log file directory. Override with <i>\$GEMSTONE_SYMBOL_GEM_LOG_DIR</i> Deleted on clean exit by default.	runsymbolgem
Admin Gem	<i>stonename_pidadmingcgem.log</i> in Stone's log file directory. Override the location with <i>\$GEMSTONE_ADMIN_GC_LOG_DIR</i> . Not deleted by default.	runadmingcgem
Reclaim Gem	<i>stonename_pidreclaimcgem.log</i> in Stone's log file directory. Override the location with <i>\$GEMSTONE_RECLAIM_GC_LOG_DIR</i> . Not deleted by default.	runreclaimcgem
RPC Gem	<i>gemnetobjectpidnode.log</i> in the <i>\$HOME</i> directory of the unix user (for gems started using the <i>gemnetobject</i> script). Override with <i>#dir</i> or <i>#log</i> in NRS of login parameters. Deleted by default on clean exit	gemnetobject (alternate scripts are also provided)
page server on remote node	<i>\$HOME/stonename_pidcachepgsvr_node.log</i> . Override with <i>#dir</i> in NRS use for the Gem login.	runcachpgsvr
page server on Stone's node for remote Gems	<i>\$HOME/runpgsvrmainpidnode.log</i> . Override with <i>#dir</i> in NRS use for the Gem login.	runpgsvrmain
pageaudit	produces both Gem and Stone logs. "Gem" log deleted by default on clean shutdown; delete behavior controlled by environment variables. "Stone" log holds audit results and is never deleted	runpageauditgem
logsender	<i>logsender_port.log</i> ; override with <i>-l</i> argument to <i>startlogsender</i> . Log file is never deleted. The same file is appended to with each restart.	startlogsender
logreceiver	<i>logreceiver_port.log</i> ; override with <i>-l</i> argument to <i>startlogreceiver</i> . Log file is never deleted. The same file is appended to with each restart.	startlogreceiver
cachewarmer (from config file)	<i>stonename_cachewarmer.log</i> in Stone's log file directory. Deleted by default on clean exit.	runcachewarmergem

Table 7.1 GemStone process types and log file details

statmonitor (from config file)	Hold statmonitor startup information; statistics are in a separate file. Deleted by default on clean exit.	runstatmonitor
--------------------------------------	--	----------------

Managing log files

Since some log files are not deleted by default, and the occasional minor error will leave log files around that would normally be deleted automatically on processes exit, the number of log files will accumulate over time. The Stone and NetLDI log files are reopened used each time the process is restarted, and are cumulative, and so these logs will grow indefinitely. So, some maintenance on GemStone log files is required. Your application's requirements for diagnostics after an incident, as well as your application design, will dictate which log files you need to retain and for how long.

Retaining or deleting all log files

The environment variables `GS_KEEP_ALL_LOGS` (page 391) and `GS_FORCE_CLEAN_LOG_FILE_DELETE` (page 391) override the individual defaults and configuration for the processes, to (respectively) force all log files to be retained, or all log files (except Stone and NetLDI) to be deleted on clean exit.

Customizing individual process deletion behavior

Many processes may have their log deletion process configured by setting the `GEMSTONE_KEEP_LOG` (page 388) environment variable in the service script that starts that process.

Refer to Table 7.1 on page 129 for specific service script names. Scripts that begin with "run", and `gemnetobject` and its variants, are found in the `$GEMSTONE/sys` directory. To configure the delete behavior:

1. make a copy of the specific script, providing your own name
2. edit the copy to set or unset `GEMSTONE_KEEP_LOG`.
3. edit `$GEMSTONE/sys/services.dat` to point the service name to your customized script. For example, if you have created a customized AdminGem script in `$GEMSTONE/scripts/myadmingcgemsript`, edit `services.dat` so the lines look something like this:

```
# runadmingcgem          $GEMSTONE/sys/runadmingcgem
runadmingcgem           $GEMSTONE/scripts/myadmingcgemsript
```

Note that customizations to scripts and `services.dat` will be lost on upgrade, and you will need to repeat this process after upgrading, to avoid the risk of missing any changes in service or script names and contents.

Localizing timestamps in log files

The timestamps printed in the log headers and in log messages are formatted according to the current system locale. You can override this using the `GS_CFTIME` environment variable. If this is set in the environment for the process, then the setting is used to control printing in log headers and log messages.

The setting for `GS_CFTIME` must be a valid `strftime` format string, and must contain fields for:

- ▶ Month: %m or %b or %B or %h
- ▶ Day: %d
- ▶ Hour: %H, or %I and %p, or %I and %P
- ▶ Minutes: %M
- ▶ Seconds: %S

If the criteria are not met, the default date format based on the system's `LOCALE` is used, or otherwise the US-centric date format.

Programmatically adding messages to logs

It may be useful for your application to deliberately write messages to the Stone or Gem logs. For example, if you are performing some automated batch processing, it may be useful to know when this started and completed in relation to other system maintenance tasks such as garbage collection.

You can write a message to the stone log using:

```
System addAllToStoneLog: aString
```

To write to the Gem log or console, you may use the following:

```
GsFile gciLogServer: aString  
GsFile gciLogClient: aString
```

Logging to the server here will write to the Gem log for an RPC session, or to the topaz console or stdout for a linked session. Logging to the client writes to the topaz console or stdout for both linked and RPC clients.

The correct place to log messages depends on your session configuration and the nature of your client application. The RPC Gem log is deleted by default if the session logs out cleanly, so any messages in it will not be retained. GUI applications may not provide access to stdout, making messages to the client inaccessible.

7.2 Repository Page and Object Audit

This section describes two levels of checks that you can perform on the repository.

- ▶ A *page audit* is invoked to ensure page-level consistency, typically after some kind of system failure, such as a read-write error or a cache coherency error. In these cases, a successful page audit indicates that the problem did not affect the committed repository. GemStone must be halted when you perform a page audit.
- ▶ An *object audit* checks the consistency of the repository at the object level. An object audit can be performed as part of routine maintenance and is performed while GemStone is running.

Page Audit

Page audits allow you to diagnose problems in the system repository by checking for consistency at the page level. Page audit can be run only on repository extents that are not in use; shut down your Stone, or make an extent copy backup.

Page audit scans the root pages in a repository, the pages used in the bitmap structures referenced by the rootpage, and all other pages (including data pages) to confirm page-level consistency. While data pages are audited, it does not check that the data on data pages is valid. For that, you need to separately run an object audit; see “Object Audit and Repair” on page 134.

To run page audit, use the **pageaudit** utility. This utility starts up an audit gem and a Stone repository monitor in audit mode, to perform the audit.

The options to pageaudit are all optional, and include:

- e *exeConfig* is the executable configuration file.
 - z *systemConfig* is the system configuration file.
 - l *logfile* is the location and name of the output file. If not specified, then the log is written to a file named *gemStoneName*-pageAudit.log in the standard Stone log file location.
 - d specifies to skip audit of data pages.
 - f specifies to keep running after an audit error is found, if possible.
 - n specifies the number of threads to use; by default, the number of extents plus the number of CPUs. Using a smaller value will cause pageaudit to take more time to complete, but reduces the impact on other processes.
- gemStoneName* is the name as which the pageaudit repository will run.; if not specified, pageaudit uses *gs64stone-pageAudit*.

The full set of options is described under “pageaudit” on page 344.

When pageaudit completes, it writes a message to stdout:

```
Page Audit of Repository completed successfully - no issues found.
For details, see /gemstone/logs/gs64stone-pageAudit.log
```

The details in the log file include Stone startup and configuration information, and audit steps performed. In addition, it produces statistics on the pages in the repository. For example:

```

PAGE AUDIT STATISTICS
RepositorySize           80.00 Mbytes      5120 pages
Data Pages               29.20 Mbytes      1869 pages
Object Table Pages       0.64 Mbytes       41 pages
Dependency Map Pages     0.02 Mbytes       1 pages
Meta Information Pages   0.38 Mbytes       24 pages
Commit RecordShadow Pages 1.09 Mbytes       70 pages
Checkpoint Shadow Pages  5.38 Mbytes       344 pages
Free Space in Repository 43.30 Mbytes      2771 pages
```

If the page audit finds problems, the message to the screen ends with a message like this:

```

----- PAGE AUDIT RESULTS -----
**** NumberOfFreePages = 980 does not agree with audit
    results = 988

**** Problems were found in Page Audit.
**** Refer to recovery procedures in System Administrator's Guide.
```

If there are problems in the page audit, you will need to restore the repository file from backups. (See the section “How to Restore from Backup” on page 219.)

Object Audit and Repair

Privileges required: SystemControl.

Object audits check the consistency of the repository at the object level. Starting with Object Table, each object is located and validated.

Object audit is performed using multiple threads (lightweight sessions), and can be configured to perform as quickly as possible using a large amount of system resources, or configured to use fewer resources and take longer to run.

Object audit should be run from linked Topaz, and on the same machine as the Stone.

```
Repository >> objectAudit
```

`objectAudit` is the normal way to perform the audit. You may have other sessions logged in and running simultaneously, but the audit will impact performance. This audit uses two threads and up to 90% of the CPU.

```
Repository >> fastObjectAudit
```

`fastObjectAudit` is like `objectAudit`, but is configured to use most or all system resources to complete as quickly as possible. This is useful when running an audit on offline systems.

```
Repository >> objectAuditWithMaxThreads: maxThreads  
percentCpuActiveLimit: aPercent
```

This method allows you to specify the exact performance/impact parameters for the audit, if neither `objectAudit` nor `fastObjectAudit` is satisfactory for your requirements.

Performing the Object Audit

To perform an object audit:

Step 1. Log in to GemStone using linked Topaz (`topaz -l`).

Step 2. Send one of the audit messages to the repository. For example:

```
topaz 1> printit  
SystemRepository objectAudit  
%
```

The audit involves a number of checks and specific error messages. Checks include:

- ▶ Object corruption – The object header should contain valid (legal) information about the object’s tag size, body size (number of instance variables), and physical size (bytes or OOPs).
- ▶ Object reference consistency – No object should contain a reference to a nonexistent object, including references to a nonexistent class.
- ▶ Identifier consistency – OOPs within the range in use (that is, up to the high-water mark) should be in either the Object Table or the list of free OOPs, and OOPs for objects existing in data pages should be in the Object Table.

If the repository is consistent and no errors are found, the audit will complete with the line:

```
Object Audit: Audit successfully completed; no errors were
detected.
```

Otherwise, the reasons for failure with the specific problems found are reported to standard output

Error Recovery

If an object audit reports errors, these issues should be addressed. You may want to contact GemStone Technical Support for advice.

The following are general approaches to errors from object audit.

Collect and reclaim garbage and retry

If errors are reported during the object audit, you may wish to perform a `markForCollection` and `reclaimAll` and repeat the object audit. This may clear up problems if the object (s) that is (are) corrupt are not referenced from any live objects. Whether this is useful will depend on the particular errors reported.

Restore from backup

The safest approach when you find object audit errors is to restore from backup. GemStone recommends that you make regular backups, run in full transaction logging mode, and archive transaction logs as needed to recover. This would allow you to recover at any time from unexpected problems such as repository corruption.

If you do not have the set of backups and transaction logs that would allow you to restore from a backup and recover later transactions, or if you are in partial transaction logging mode, you can still make and restore a backup. Backups made using `fullBackupTo:`, when restored, rebuild the internal data structures. Depending on the specific problems found in audit, this may clear up the problem.

Attempt repair

GemStone includes the ability to repair invalid references, but this can only repair detectable corruption. If there are a number of errors reported, whatever caused the objects to disappear or become invalid may easily have also introduced undetectable logical corruption.

However, a single invalid reference may not indicate a widespread problem, and repair may allow important data to be recovered.

To manually repair an individual invalid reference, use the Topaz object specification format `@identifier` to substitute nil or an appropriate reference for an invalid reference.

For example, given an instance of `Array` with the OOP 51369729, if the element at slot 3 is an object that does not exist, it can be repaired by setting the reference to nil using the following expression:

```
topaz 1> send @51369729 at: 3 put: nil
```

The method `Repository >> repair` will perform an audit and make repairs during the re-scan. The following repairs are done:

- ▶ nil is substituted for an invalid object reference.
- ▶ Class `String` is substituted for an invalid class of a byte object, class `Array` for a pointer object, or class `IdentitySet` for a nonsequenceable collection object.

- ▶ Oops in the Object Table for which the referenced object does not exist are inserted into the list of free Oops.
- ▶ Oops for which an object exists but which are also in the list of free Oops are removed from the free list.

The repair audits the repository, keeping track of errors. After the initial audit completes, each error found is repaired. A descriptive message is displayed for each repair. The repair will commit periodically to avoid memory issues, and log off when it is complete.

7.3 Profiling Repository Contents

Some questions, such as “what is using up all the space in my Repository?”, can only be answered by examining the types and numbers of objects in your repository. To find out this information, you can use methods on `GsObjectInventory`.

The methods in `GsObjectInventory` count all instances of all classes in the repository – or in any collection, or in a hidden set, or in a file of disconnected possible garbage objects – and report the results, ordered by the number of instances or by space consumed.

`GsObjectInventory` performs a multi-threaded scan of the repository, and thus should only be run in session on the same machine as the Stone. To tune the impact of the scan, additional protocol allows you to perform fast scans or to specify the impact levels. For details, see methods in the image.

The following code will report the number of instances and the space required for all Classes whose total space requirements are more than 50000 bytes.

Example 7.2 Object Inventory byteCountReport

```

topaz 1> run
GsObjectInventory profileRepository byteCountReportDownTo: 50000
%
*** GsObjectInventory byteCountReport printed at: 19/09/2017
10:54:49 ***
Hidden classes are included in this report.

```

Class	Instances	Bytes
String	32291	8263560
GsNMethod	23113	4628608
Array	26775	4273072
GsMethodDictionary	3844	1963336
Symbol	20253	909944
CanonStringBucket	2019	307888
Class	1888	294800
IdentityKeyValueDictionary	1913	260216
SymbolAssociation	5525	221584
ExecBlock	3212	205768
LargeObjectNode	16	199072
SymbolDictionary	991	165584
SymbolSet	5081	159200
IdentityCollisionBucket	1275	136408

The same profiling with an instance count report is much shorter, since the number of instances, rather than the bytes of space used, limits the results.

Example 7.3 Object Inventory instanceCountReport

```

topaz 1> run
GsObjectInventory profileRepository instanceCountReportDownTo:
10000
%
*** GsObjectInventory instanceCountReport printed at: 19/09/2017
11:02:01 ***
Hidden classes are included in this report.

```

Class	Instances	Bytes
String	32291	8263560
Array	26775	4273072
GsNMethod	23113	4628608
Symbol	20253	909944

Both of these reports include instances of hidden classes, classes that are used to implement internal GemStone objects, which are invisible to the image. One such class is `LargeObjectNode`. Instances of `LargeObjectNodes` are used to implement the tree structures that underlie large collections. To avoid seeing hidden classes, profile using the method `profileRepositoryAndSkipHiddenClasses`.

For more on `GsObjectInventory`, see the methods in the image.

7.4 Monitoring Performance

As part of your ongoing responsibilities, you may find it useful to monitor performance of the object server or individual session processes.

GemStone includes graphical tools to allow you to record statistics in file and analyze this data graphically. You can also programmatically access these statistics.

A full list of the statistics that are recorded and are available programmatically can be found in the *VSD User's Guide*.

Statmonitor and VSD

GemStone includes the **statmonitor** utility, which records statistics about GemStone processes to a disk file. You can configure the processes for which statistics recorded, how frequently the statistics are collected, and other details. See **statmonitor** on page 357 for more information.

Both GemStone-specific and operating system statistics are collected. The operating system statistics include general host information as well as information specific to the individual GemStone processes.

We recommend running `statmonitor` at all times, as it provides a valuable record of many aspects of system behavior. If you encounter certain kinds of problems in your application, GemTalk Technical Support will request `statmonitor` data for the period leading up to the problem, to diagnose possible causes.

You can configure `statmonitor` to start automatically on stone startup using the `STN_STATMONITOR_ARGS` configuration option (page 328). Similar options allow you to automatically start `statmonitor` on node that run a remote or mid-level shared page cache.

The data generated by `statmonitor` is normally viewed graphically using VSD (Visual Statistics Display). For details on using VSD, see the *VSD User's Guide*.

Programmatic Access to Cache Statistics

A set of methods on the `System` class provide a way for you to analyze performance by programmatically examining the statistics that are collected in the shared page cache. This is the same data that is visible using `statmonitor` and VSD, although `statmonitor` and VSD can collect additional OS level information. This additional OS level information is also available programmatically; see "Host Statistics" on page 143

A process can only access statistics that are kept in the shared page cache to which it is attached. Sessions that are running on a different node than the Stone use a separate shared cache on that remote node. This means that processes that are on a different node than the Stone, cannot access statistics for the Stone or for other server processes that are attached to the Stone's shared page cache.

Within the shared page cache, GemStone statistics are stored as an array of *process slots*, each of which corresponds to a specific process. Process slot 0 is the shared page cache monitor. On the Stone's shared page cache, process slot 1 is the Stone; on remote caches, slot 1 is the page server for the Stone that started the cache. Subsequent process slots are the page servers, Admin and Reclaim Gems, Symbol Gem, and user Gems. The order of these slots depends on the order in which the processes are started up, and is different on remote caches.

The specific set of statistics is different for each type of process that can attach to the shared page cache. The types of processes are numbered:

- 1 = Shared page cache monitor
- 2 = Stone
- 4 = Page server
- 8 = Gem (including Topaz, GBS, and other GCI applications).
- 16 = Statmonitor

Further numbers includes those for shared counters, platform-specific OS system stats, and so on.

Statistics by name

To obtain the value for a specific statistics for the Stone, the Stone's SPC monitor, or for the current session, use the following methods:

```
System class >> stoneCacheStatisticWithName:
System class >> primaryCacheMonitorCacheStatisticWithName:
System class >> myCacheStatisticWithName:
```

These methods will return the statistics value corresponding to the given name for that process. If the statistics name is not found, it returns nil.

For example, to retrieve the statistics named 'CommitRecordCount' for the Stone:

```
topaz 1> printit
System stoneCacheStatisticWithName: 'CommitRecordCount'.
%
23
```

To retrieve the current session's PageReads:

```
topaz 1> printit
System myCacheStatisticWithName: 'PageReads'.
%
548
```

All statistics for a process

The general way to retrieve statistics is as an array of values. To understand what the value at each index refers to, there are corresponding description methods to return an array of Strings. Matching the index of the statistic name to the index within the values locates the value for that statistic.

Since the statistics are different for the different types of processes, you will need to use corresponding methods to collect the statistics and the descriptions.

For the Stone, the Gem that is running the code, and the Stone's shared page cache monitor, no further information is needed to identify them within the cache, so the following pairs of methods can be used:

```
System cacheStatisticsDescriptionForGem.
System myCacheStatistics.

System cacheStatisticsDescriptionForStone.
System stoneCacheStatistics.

System cacheStatisticsDescriptionForMonitor.
System sharedPageCacheMonitorCacheStatistics.
```

For example, while you would normally use `stoneCacheStatisticForName:`, here is another possible way to get the `CommitRecordCount`:

```
topaz 1> printit
| index |
index := System cacheStatisticsDescriptionForStone
        indexOf: 'CommitRecordCount'.
System stoneCacheStatistics at: index.
%
23
```

To collect statistics for other Gems, and for page servers, you need to determine the process Id, session Id, or slot of the specific Gem or page server, or the cache name of the Gem. There are a variety of ways you might determine this, but one way is to examine the results of:

```
System cacheStatisticsForAllSlotsShort
```

This method returns the name, process Id, session Id, statistics type, and process slot for each process currently attached to the cache. For example:

```
topaz 1> printit
(System cacheStatisticsForAllSlotsShort) collect:
[:ea | ea printString]
%
an Array
#1 anArray( 'ShrPcMonitor', 7722, 4294967295, 1, 0)
#2 anArray( 'gs64stone', 7721, 0, 2, 1)
#3 anArray( 'FreeFrmPgsvr2', 7725, 4294967294, 4, 2)
#4 anArray( 'AioPgsvr3', 7726, 4294967294, 4, 3)
#5 anArray( 'pagemgrThread', 7729, 1, 8, 4)
#6 anArray( 'GcAdmin5', 7734, 2, 8, 5)
#7 anArray( 'SymbolGem6', 7735, 3, 8, 6)
#8 anArray( 'GcReclaim6_7', 7733, 4, 8, 7)
#9 anArray( 'Gem26', 2271, 5, 8, 8)
#10 anArray( 'Gem27', 16924, 6, 8, 9)
```

Of course, a Gem may log out between the time you execute this and the time you collect statistics, so be sure that your code handles that condition gracefully.

The methods you use to get the statistics and the corresponding descriptions will depend on how you have determined the specific process you want information about.

By name:

```
System cacheStatisticsForProcessWithCacheName: aString
(You must manually determine the process type)
```

or

```
System cacheStatsForGemWithName: aString.
System cacheStatisticsDescriptionForGem.
```

By operating system Process Id (PID):

```
System cacheStatisticsProcessId: aPid.
System cacheStatisticsDescriptionAt:
(System cacheSlotForProcessId: aPid).
```

By process slot:

```
System class >> cacheStatisticsAt: aProcessSlot
System class >> cacheStatisticsDescriptionAt: aProcessSlot
```

By session Id:

The page server for a Gem assumes the same sessionId as its Gem.

```
System gemCacheStatisticsForSessionId: aSessionId.
System cacheStatisticsDescriptionForGem.
```

or

```
System cacheStatsForPageServerWithSessionId: aSessionId
System cacheStatisticsDescriptionForPageServer
```

For example, to find an aggregate value for TimeInFramesFromFindFree of all Gems in the system:

```
topaz 1> printit
| gemPids index time |
gemPids := Array new.
System cacheStatisticsForAllSlotsShort do:
    [:anArray |
    (anArray at: 4) = 8 ifTrue:
        [gemPids add: (anArray at: 2)].
    ].
index := System cacheStatisticsDescriptionForGem indexOf:
    'TimeInFramesFromFindFree'.
time := 0.
gemPids do: [:aPid | | stats |
    stats := System cacheStatisticsProcessId: aPid.
    stats ifNotNil: [time := time + (stats at: index)].
    ].
time
%
```

Setting the name for the Gem in the cache

To make it easier for you to track cache statistics for specific Gems, you can explicitly give each Gem a unique name. The method

```
System cacheName: aString
```

sets the name for the current Gem session in the cache statistics, thus making it much easier to read the statistics in VSD.

When using `topaz`, the `-u` command line argument will set the cachename for logins from `topaz`.

Otherwise, set the cache name soon after login. If you are collecting statistics information using `statmonitor`, information may be logged using the default name for the Gem when the session first logs in, in which case so you may have two separate lines of data for the same session.

Session Statistics

In addition to the system-generated statistics listed below, GemStone provides a facility for defining session statistics — user-defined statistics that can be written and read by each session, to monitor and profile the internal operations specific to your application.

There are 48 session cache statistic slots available, with names of the form `SessionStat01...SessionStat47`.

You can use the following methods to read and write the session cache statistics:

```
System class >> sessionCacheStatAt: anIndex
  Returns the value of the statistic at the designated index. anIndex must be in the range
  -2 to 47. Negative indexes are reserved for internal use.
```

```
System class >> sessionCacheStatAt: anIndex put: aValue
  Assigns a value to the statistic at the designated index and returns the new value.
  anIndex must be in the range -2 to 47. Negative indexes are reserved for internal use.
```

```
System class >> sessionCacheStatAt: anIndex incrementBy: anInt
  Increment the statistic at the designated index by anInt, and returns the new value.
  anIndex must be in the range -2 to 47. Negative indexes are reserved for internal use.
```

```
System class >> sessionCacheStatAt: anIndex decrementBy: anInt
  Decrement the statistic at the designated index by anInt, and returns the new value.
  anIndex must be in the range -2 to 47. Negative indexes are reserved for internal use.
```

```
System class >> sessionCacheStatsForProcessSlot: aProcessSlot
  Return an array containing the 48 session statistics for the given process slot, or nil if
  the process slot is not found or is not in use.
```

```
System class >> sessionCacheStatsForSessionId: aSessionId
  Return an array containing the 48 session statistics for the given session id, or nil if the
  session is not found or is not in use.
```

Global Session Statistics

In addition to the Gem session statistics, GemStone/S 64 Bit provides global session statistics — user-defined statistics that can be written and read by any Gem on any Gem server. Unlike *session* cache statistics, which are stored in the shared page cache of the machine that the Gem is running on, *global* session statistics are stored in the shared page cache of the Stone. Global session statistics are not transactional. For a given statistic, every session sees the same value, regardless of its transactional view.

There are 48 global cache statistic slots available, with names of the form `GlobalStat00...GlobalStat47`.

You can use the following methods to read and write the global cache statistics:

```
System class >> globalSessionStatAt: aProcessSlot
  Returns the value of the statistic at the designated slot (must be in the range 0..47).
```

System class >> globalSessionStatAt: *aProcessSlot* put: *aValue*
 Assigns a value to the statistic at the designated slot (must be in the range 0..47) and returns the new value. The value must be a SmallInteger in the range of -2147483648 to 2147483647.

System class >> incrementGlobalSessionStatAt: *aProcessSlot* by: *anInt*
 Increments the value of the statistic at the designated slot by *anInt* and returns the new value of the statistic. The value *anInt* must be a SmallInteger in the range of -2147483648 to 2147483647.

Host Statistics

Host Statistics for processes

Process-level statistics require an OS call, which can cause cache statistics to impact performance. These statistics are not part of the information returned by regular cache statistics interface methods. To get this information, use the following methods.

System class >> hostProcessStatisticsNames
 Returns an array of Strings which are the names of the per-process statistics provided by this host.

System class >> hostStatisticsForMyProcess
 Returns an array of SmallIntegers which represent the host statistics for this process. The names of each statistic are returned by the #hostProcessStatisticsNames method.

System class >> hostStatisticsForProcess: *processId*
 Returns an array of SmallIntegers which represent the host statistics for the process with the given process ID. The names of each statistic are returned by the #hostProcessStatisticsNames

Specific methods are also available to return the host CPU statistics only:

System class >> hostCpuStatsForProcessId: *anInt*
 Return an Array of two integers as follows:

- 1 - user mode CPU milliseconds
- 2 - system mode CPU milliseconds

Both array elements will be -1 if the process slot is out of range or not in use or if this method is not supported for the host architecture.

It is not required that the process with pid *anInt* is attached to the shared page cache or even is a GemStone process. The method will succeed for any process for which the Gem session executing the method has permission to view the target process' CPU usage statistics.

System class >> hostCpuStatsForProcessSlot: *anInt*
 For the process using the cache process slot *anInt*, return an Array of two integers as follows:

- 1 - user mode CPU milliseconds used
- 2 - system mode CPU milliseconds used

Both array elements are set to -1 if the process slot is out of range or not in use, or if this method is not supported for the host architecture.

Host Statistics for OS

While most monitoring is of the object server and session processes, it is also useful to monitor the performance of the operating system that is running GemStone. On host platforms that support it, the following methods return statistics provided by the operating system. This is the same information that is available via statmonitor; see **statmonitor** on page 357.

`System class >> fetchSystemStatNames`

Return an array of Strings with the names of the available OS level statistics. The length is host-dependent. If the host system does not support system statistics, this method returns nil.

`System class >> fetchSystemStats`

Return an array of Numbers corresponding to the names returned by the `#fetchSystemStatNames` method. The length of the result array is host dependent. While most elements in the result array will be SmallIntegers, the result may also contain other types of Numbers such as SmallDoubles, Floats, LargeIntegers, etc. If the host system does not support system statistics, this method returns nil.

You can also monitor specific CPU usage for the host using the following method:

`System class >> hostCpuUsage`

Returns an Array of 5 SmallIntegers with values between 0 and 100 which have the following meanings:

- 1 - Percent CPU active (user + system)
- 2 - Percent CPU idle
- 3 - Percent CPU user
- 4 - Percent CPU system (kernel)
- 5 - Percent CPU I/O wait

On hosts with multiple CPUs, these figures represent the average across all processors. The results of the first call to this method are invalid and should be discarded. Returns nil if the host system does not support collecting CPU statistics.

User Accounts and Security

This chapter also shows you how to perform some common GemStone user administration tasks:

- ▶ How to create and modify user accounts, including privileges, group memberships, symbol resolution, and how to control the user's read-write access to objects through the use of object security policies.
- ▶ How to set up login authentication, including GemStone authentication as well as external authentications, such as by UNIX `userId`, LDAP, or Kerberos.
- ▶ Tracking user account logins and logouts

To perform most of these tasks you must have explicit *privilege* to execute a restricted Smalltalk method, and you may also need to be explicitly authorized to modify an affected `GsObjectSecurityPolicy`. This chapter describes how users can be configured with the appropriate privileges and object security policies. To understand how `GsObjectSecurityPolicies` can be used to control access to data, see the chapter "Object Security and Authorization" in the *Programming Guide*.

8.1 GemStone Users

This section provides background information about how GemStone stores user accounts, predefined system users, groups of users, and users's name space.

UserProfiles

Each GemStone user is associated with an instance of class `UserProfile`. This `UserProfile` object contains information describing objects that the user is allowed to examine or modify, privileges that the user has to perform certain operations, and security information.

Each UserProfile has the following information:

User ID	A unique String that identifies the user to the GemStone system.
Authentication Scheme	How this user is authenticated; #GemStone, #UNIX, #LDAP, or #SingleSignOn.
Password	The GemStone-specific password (an InvariantString) to use to validate logins for #GemStone authentication. GemStone stores the password in encrypted form in a secure manner.
Default Object Security Policy	Either nil or an instance of GsObjectSecurityPolicy. This determines the default read and write authorizations for objects created by the user.
Privileges	Encoding for a logical collection of symbols that allow the user to perform certain “privileged” system functions.
Groups	In conjunction with object security policies, group membership is used to allow access to restricted objects for specific categories of users.
SymbolList	The list of SymbolDictionaries that is used to resolve references to Classes and other Globals for this user.
Login Hook	Method selector or block of code to execute on login.

These are discussed in more detail under “UserProfile Data” on page 147.

Other information related to the user account is stored in an instance of UserSecurityData; this includes data related to security features. Instances of UserSecurityData are private and protected, but some information, such as lastLoginTime, can be accessed via methods in UserProfile.

AllUsers

Each instance of UserProfile must be in the global collection, AllUsers. AllUsers is the single instance of UserProfileSet. AllUsers acts as the “root” for all objects in the repository; any object in the repository must be reachable from AllUsers, usually via the SymbolLists of the UserProfiles, otherwise it is subject to garbage collection.

Special System Users

When GemStone is first installed, AllUsers has UserProfiles already defined for the following users. These are the special system users. *You must never delete these users.* These users may not have privileges removed, cannot be disabled, must use GemStone authentication, and their accounts are not subject to password or account age limits.

You can determine if an account is a special system user by executing:

```
UserProfile isSpecialUserId: 'theUserId'
```

SystemUser

SystemUser is analogous to root in UNIX. SystemUser has all privileges, belongs to all predefined groups, and is authorized to read and write all objects regardless of

GsObjectSecurityPolicy protection. These privileges cannot be taken away, so SystemUser can always write to all objects. This account is used only to perform GemStone system upgrades, modify some system configuration settings, and other special-purpose operations that must be highly restricted.

SystemUser can only be configured to use GemStone authentication and cannot be disabled.

The SystemUser account is the owner of the SystemObjectSecurityPolicy, which contains the kernel classes.

WARNING

*Logging in to GemStone as SystemUser is like logging in to your workstation as root: an accidental modification to a kernel object can cause a great deal of harm. Use the DataCurator account for system administration functions except those that **require** SystemUser privileges, such as a repository upgrade.*

DataCurator

The DataCurator account is the account that is normally used for day-to-day administration tasks. Initially, DataCurator is granted all privileges and belongs to all predefined groups.

DataCurator can be configured to use GemStone or SingleSignIn authentication, but not LDAP or UNIX, and cannot be disabled.

All GemStone UserProfiles are protected by the DataCuratorObjectSecurityPolicy.

GcUser

The GcUser account is a special account that logs in to the repository automatically to perform garbage collection tasks. You normally would only login as GcUser in order to update configuration parameters stored in GcUser's UserGlobals.

GcUser can be configured to use GemStone or SingleSignIn authentication, but not LDAP or UNIX, and cannot be disabled.

SymbolUser

The SymbolUser account is a special account that is used to perform symbol creation tasks. Login as SymbolUser is disallowed. Directly accessing the AllSymbols collection, which is in the UserGlobals of the SymbolUser, is possible from another administrative session.

Nameless

The Nameless account is a special account for use only by other GemStone products. Do not use this account or change it unless instructed to do so by GemStone Technical Support.

UserProfile Data

Each user profile contains important information about the account, which may be explicitly specified during instance creation or rely on default values. This information may also be updated during the course of time for the UserProfile.

The requirements for updating this information vary. In many cases, the requirements are different between updating information for another user and for updating your own information. See the update methods for details.

User ID

Each UserProfile is created with a unique user Id String. Embedded spaces are permitted, and characters in the byte range (up to codePoint 255) are allowed.

UserId may only be changed by SystemUser or by a user with the privilege #ChangeUserId. The userIds of special system users cannot be changed.

Authentication Scheme

When a user wants to log in to the GemStone repository, their login must be authenticated: they must present a password that is matched against stored information for their UserId, to verify that they are authorized to log in. This authentication can be done entirely within GemStone, or GemStone can use UNIX, LDAP, or Kerberos to perform the authentication.

Performing the authentication entirely within GemStone – Gemstone authentication – is the initial default for all users. Other authentication schemes can be configured for individual UserProfiles, though there are restrictions for system accounts. The repository may contain UserProfiles using a mix of authentication schemes.

After the authentication scheme is modified for a user and the change is committed, it will take effect the next time the user logs in. Existing logins are not affected.

In addition to authentication for the GemStone UserProfile, users may need to authenticate the UNIX username, before the NetLDI will start the Gem process for an RPC login. This depends on how the system is configured; see Chapter 4 for details on interprocess security and host authentication.

Changing authentication, or inquiring about authentication, requires #OtherPassword privilege.

Setting up GemStone and other authentication schemes is described in more detail starting on page 166.

Determining an Account's Authentication Scheme

Your repository may contain a mix of authentication schemes, with some users (such as SystemUser) using GemStone authentication, others authenticating using UNIX, LDAP or SingleSignOn. You can determine what scheme an account is using by sending authenticationScheme. This will return the symbol #GemStone, #UNIX, #LDAP, or #SingleSignOn. For example,

```
(AllUsers userWithId: 'DataCurator') authenticationScheme.  
%  
GemStone
```

Password

Passwords for GemStone authentication must be:

- ▶ invariant Strings
- ▶ not empty
- ▶ be different than the User Id
- ▶ no longer than 1024 characters
- ▶ contain only Characters with codePoints under 256

When each UserProfile is created, the initial password is defined and the account starts off using GemStone authentication. If the UserProfile is modified to use another

authentication scheme, the initial password is discarded. For GemStone authentication, the password supplied in the login parameters is verified against this password.

GemStone authentication provides a number of controls on passwords, which apply to changing passwords and constraints on choice of password.

Default Object Security Policy

A user's defaultObjectSecurityPolicy determines the default read and write authorizations for objects created by the user. When you add a new user to the GemStone system, you can either allow the default security policy to be nil, use protocol that creates a new GsObjectSecurityPolicy, or specify an existing GsObjectSecurityPolicy for the user.

For more information on how security policies are used to control read and write authorization for objects in the repository, see the chapter in the *Programming Guide* that discusses security.

A defaultObjectSecurityPolicy of nil means that objects created by that user, by default, have world read and write access; that is, are not restricted from being read or written by all other users. Not requiring authorization checks has the benefit of improved performance, if your application does not require object level security.

When creating a user, you can use methods that specify that a new instance of GsObjectSecurityPolicy should be created and assigned as the default for the new user. Otherwise, you must use an already committed instance of GsObjectSecurityPolicy, either existing or newly created, or nil.

If the defaultObjectSecurityPolicy for a user is not nil, the user **MUST** have write authorization to this security policy; otherwise, this user will not be able to log in.

To modify your own defaultObjectSecurityPolicy, you must have the #DefaultObjectSecurityPolicy privilege. To modify the defaultObjectSecurityPolicy of another user, you must have this privilege, and write access to the security policy of that UserProfile.

Privileges

When you create a new UserProfile, you determine whether the new user may perform certain "privileged" system functions. For example, stopping another session, or the repository itself, requires a particular privilege to do so. Table 8.1 describes the types of functions that each privilege controls.

Note that privileges are more powerful than security policy authorization. Although the owner of a security policy can always use authorization protocol to restrict read or write access to objects in a policy, an administrator with appropriate privileges, such as DataCurator, can override that protection by sending privileged messages that let you change the authorization scheme.

Table 8.1 GemStone Privileges

Type of Privilege	Privileged Operations
SystemControl	SystemControl is required by methods that start or stop sessions, including operations that invoke the Multi-Threaded Scan (page 278); for methods that suspend or resume logins, send signals to other sessions, and manage checkpoints.

Table 8.1 GemStone Privileges (Continued)

Type of Privilege	Privileged Operations
SessionAccess	SessionAccess privilege is required to find out information about sessions other than the current session, or to perform operations on other sessions.
UserPassword	Required to change your own password using <code>UserProfile>>oldPassword:newPassword:</code>
DefaultObjectSecurityPolicy	This privilege is required to set a UserProfile's default ObjectSecurityPolicy using <code>UserProfile>>defaultObjectSecurityPolicy:</code> or a method that invokes that. For compatibility with previous versions, the DefaultSegment privilege also resolves to this privilege
CodeModification	You must have CodeModification privilege to create or modify instances of GsNMethod, GsMethodDictionary, or Class. See the discussion following this table.
OtherPassword	You must have OtherPassword privilege to make any changes to a UserProfile other than your own. This includes adding or removing a SymbolDictionary to/from a SymbolList that is not your own. OtherPassword is also required to find out information about UserProfiles other than the currently logged in session. OtherPassword is required to make any changes to AllUsers, including creating a new user and configuring security requirements.
ObjectSecurityPolicy Creation	Required in order to creating a new GsObjectSecurityPolicy, using <code>GsObjectSecurityPolicy class>>new, newInRepository:</code> or any methods that invoke these. For compatibility with previous versions, the SegmentCreation privilege also resolves to this privilege.
ObjectSecurityPolicy Protection	You must have ObjectSecurityPolicyProtection to update the authorizations of a GsObjectSecurityPolicy, other than one that is owned by the current session's user. This includes <code>GsObjectSecurityPolicy>>group:authorization:, ownerAuthorization:, and worldAuthorization:.</code> For compatibility with previous versions, the SegmentProtection privilege also resolves to this privilege.
FileControl	FileControl is required for system operations that access external files, including operations related to backup, restore, transaction logs, and extents. This does not affect application access using GsFile.
GarbageCollection	Required to perform any garbage collection operation, to start and stop Admin and Reclaim Gems, and force epoch or reclaim to run. Also required to audit and profile the repository.
NoPerformOnServer	If you have this privilege, you cannot execute <code>System class>>performOnServer:</code>

Table 8.1 GemStone Privileges (Continued)

Type of Privilege	Privileged Operations
NoUserAction	If you have this privilege, you cannot execute <code>System class>> loadUserActionLibrary:</code>
NoGsFileOnServer	If you have this privilege, you cannot execute any GsFile operation which accesses a file on the server.
NoGsFileOnClient	If you have this privilege, you cannot execute any GsFile operation which accesses a file on the client.
SessionPriority	Required to modify the priority of any session, or to check the priority of a session other than the current session.
CompilePrimitives	Allow user to compile primitive methods, which is otherwise restricted to SystemUser.
ChangeUserId	Allow user to execute <code>userId:password:</code> to rename a user, which is otherwise restricted to SystemUser.

Code Modification Privilege

CodeModification privilege is required to execute any method that modifies Smalltalk code. This privilege is required for all developers writing code.

- ▶ You must have #CodeModification privilege to create instances of GsNMethod, or to create or modify instances of GsMethodDictionary or Class. (You cannot modify a GsNMethod once it has been created.)
- ▶ You must have #CodeModification privilege to add a Class to, or remove a Class from, a SymbolDictionary or its subclasses.
- ▶ You must have #CodeModification privilege to add or remove a SymbolDictionary from your own SymbolList.

You cannot use GemBuilder for C to modify instances of the following classes (or their subclasses): GsNMethod, GsMethodDictionary, Class, SymbolDictionary, SymbolList, UserProfile.

Symbol Lists

As explained in the *Programming Guide*, the GemStone Smalltalk compiler follows a well-defined path in resolving objects named by source code symbols. First, the compiler considers the possibility that a variable name might be either local (a temporary variable or an argument) or defined by the class of the current method definition (an instance variable, a class variable, or a pool variable). If a variable is none of these, the compiler refers to an Array of SymbolDictionaries in the user's UserProfile and current session state. That Array is called the user's *symbol list*. The symbol list tells Smalltalk which of many possible GemStone SymbolDictionaries to search for an object named in a Smalltalk program.

For each user, a persistent instance of class SymbolList is stored in the repository and is referenced from the UserProfile associated with this user as the symbolList instance variable. In addition, a transient copy of that SymbolList is stored in the GsCurrentSession object for the logged-in session.

A session's transient copy can be modified without affecting (or causing concurrency conflicts with) either the persistent symbol list or the transient copies controlling other sessions. Changes to your own UserProfile's persistent symbol list also change the symbol resolution of your current session. However, changes to the persistent symbol list are likely to cause concurrency conflicts with other sessions logged in under the same userId.

For further information about symbol lists, refer to the *Programming Guide*.

New UserProfiles are created with the following SymbolDictionaries, in this order:

UserGlobals	Each UserProfile has its own SymbolDictionary for the user's private symbols.
Globals	The second element in each user's initial symbol list is a "system globals" SymbolDictionary, <i>Globals</i> . This dictionary contains all of the GemStone Smalltalk kernel classes (Object, Class, Collection, Integer, and so forth). Although users can read the objects in Globals, ordinarily they cannot modify objects in that Dictionary.
Published	The third and final element in each user's initial symbol list is a SymbolDictionary for application objects that are "published" to all users. Users who are members of the group Publishers can place objects in this dictionary to make them visible to other users. Using the Published dictionary lets you share these objects without having to put them in Globals, which contains the GemStone kernel classes, and without the necessity of adding a special dictionary to each user's symbolList instance variable.

Although all users automatically share access to objects in Globals, sharing application objects between users requires that the objects be in a SymbolList that is visible to both users. There are three primary ways to do this:

- ▶ As a member of group Publishers, you can add the objects to the Published dictionary. This dictionary is already in each user's symbol list, so whatever you add becomes visible to users the next time they obtain a fresh transaction view of the repository. You may do this by sending the message `Published at: aKey put: aValue`.
- ▶ You can define a special SymbolDictionary, and add that to the user's SymbolList. The procedure is described under "Adding a SymbolDictionary to Someone Else's Symbol List" on page 162.
- ▶ The application itself can add the objects to the individual user's symbol list, either to the permanent symbol list in the UserProfile or to a transient symbol list for that session. For information about this approach, refer to the *Programming Guide*.

For more information, refer to the chapter on symbol resolution and object sharing in the *Programming Guide*.

DeletedUserProfile and AllDeletedUsers

Removing a user requires some cleanup to ensure that the references to or from the UserProfile do not allow premature garbage collection of objects that are needed, or prevent garbage collection entirely. There are two kinds of references of concern:

- ▶ references from an instance of GsObjectSecurityPolicy to the user, which can prevent the userProfile from being garbage collected
- ▶ references from the userProfile to the symbol list and to objects that are private to the user that is being deleted, which may need to be retained.
- ▶ references from UserProfileGroups

To avoid the risk of problems, removing a user requires using specific protocol that performs cleanup.

These methods update any GsObjectSecurityPolicies to be owned by either the current user or a specified user, and removes the user from each group in AllGroups. To ensure that objects whose only reference is from the deleted user are not prematurely lost, when the instance of UserProfile is deleted a new instance of DeletedUserProfile is created, and added to the globals AllDeletedUsers. This DeletedUserProfile has a reference to the SymbolList of the deleted user, as well as the userId and the date the user was deleted.

An administrator should review the classes and methods in the SymbolLists of the DeletedUserProfile, copy anything valuable elsewhere, and manually remove the DeletedUserProfile from AllDeletedUsers.

8.2 UserProfileGroups

GemStone uses a combination of group memberships and GsObjectSecurityPolicy group permissions to control access to groups of objects by groups of users.

UserProfileGroups are also used to handle KerberosPrincipals that will be shared by multiple userProfiles for #SingleSignOn authentication.

A UserProfileGroup maps a group name to a set of UserProfiles. For compatibility with past releases, protocol in UserProfile and GsObjectSecurityPolicy may accept arguments of the group name instead of, or in addition to, the UserProfileGroup instance.

AllGroups

AllGroups is a global collection of UserProfileGroups, that includes all groups defined for users and security policies.

Initially, AllGroups contains the following predefined groups:

Table 8.2 GemStone Groups

Group name	Access
System	Members of this group have write access to objects protected by the GcUser's object security policy.
DataCuratorGroup	Members of this groups have write access to objects protected by the DataCuratorObjectSecurityPolicy . This is useful if you wish to make a user other than DataCurator to be a system administrator, since many operations that update users require write access to DataCuratorObjectSecurityPolicy .
Publishers	Members of this group have write access to objects protected by PublishedObjectSecurityPolicy .

Table 8.2 GemStone Groups

Group name	Access
Subscribers	Members of this group have read access to objects protected by PublishedObjectSecurityPolicy .
SymbolUser	(Reserved for future use).

By default, all new users become members of group Subscribers.

Groups for object authorization

It is common that certain objects must be protected from read or write access by other users in the system (the “world”), while still being accessible to specific individual users. By creating a group, adding authorization for that group to the `GsObjectSecurityPolicy` that protects these objects, and by making the user a member of the group, you can provide that user with the appropriate access to these objects.

A `GsObjectSecurityPolicy` can authorize multiple groups and a user can be a member of multiple groups.

Create a group

To create a new group and add it to `AllGroups`, use:

```
UserProfileGroup class >> newGroupWithName:
```

To lookup an existing group by name, execute one of:

```
UserProfileGroup class >> groupWithName:
```

```
UserProfileGroup class >> groupWithName:ifAbsent:
```

```
UserProfileGroup class >> groupWithName:otherwise:
```

Delete a group

Deleting a group removes it from all `User` accounts that are members, as well as from `AllGroups`.

```
UserProfileGroup class >> deleteGroup:
```

```
UserProfileGroup class >> deleteGroupWithName:
```

```
UserProfileGroup class >> deleteGroupWithName:ifAbsent:
```

8.3 Creating and Removing Users

Methods that create `UserProfiles` add the new `UserProfile` to `AllUsers`, the global collection or users (a singleton instance of `UserProfileSet`). `UserProfiles` that are not in `AllUsers` cannot log in.

In addition to creating the new `UserProfile`, you should also ensure that each user’s UNIX environment is set up to provide access to GemStone. This is described in the *GemStone/S 64 Bit Installation Guide*.

Removing a user, in addition to removing the `UserProfile` from `AllUsers`, requires cleanup to ensure that objects that refer to the deleted user do not inadvertently prevent the deleted

user from being garbage collected, and that objects that are referred to only by the deleted users are not inadvertently garbage collected. The methods to remove users perform this cleanup, creating an instance of DeletedUserProfile in the AllDeletedUsers global.

Creating Users

Privileges required: OtherPassword, and write access to the DataCuratorObjectSecurityPolicy. You may also need ObjectSecurityPolicyCreation.

Simple User Creation

At minimum to create a new UserProfile, you must supply the new user's userId and password, each as a String. This creates the new user with no privileges, only the Subscribers group, and with a nil defaultObjectSecurityPolicy.

```
AllUsers addNewUserWithId: 'theUserId'
          password: 'thePassword' .
```

Simple User Creation with GsObjectSecurityPolicy Creation

To create a new user and specify that a new instance of GsObjectSecurityPolicy should be created for the new user, use the following expression:

```
AllUsers addNewUserWithId: 'theUserId'
          password: 'thePassword'
          createNewObjectSecurityPolicy: true
```

User Creation With Privileges, Groups, and ObjectSecurityPolicy

Using the complete form allows you to assign privileges to the new user, add the user to one or more groups, and specify a default ObjectSecurityPolicy. The ObjectSecurityPolicy may be nil.

```
AllUsers addNewUserWithId: 'theUserId'
          password: 'thePassword'
          defaultObjectSecurityPolicy: anObjectSecurityPolicyOrNil
          privileges: anArrayOfPrivSyms
          inGroups: aCollectionOfGroupsOrGroupNames
```

For example:

```
topaz 1> printit
AllUsers addNewUserWithId: 'Mary'
          password: 'herPasswd'
          defaultObjectSecurityPolicy: nil
          privileges: #( UserPassword )
          inGroups: #( 'MarathonRunners' ).
System commitTransaction.
%
```

For additional user creation protocol, see the image. UserProfileSet instance methods and UserProfile class methods both create new UserProfiles and add the new user to AllUsers.

Removing Users

Privileges required: OtherPassword.

In addition to removing the UserProfile from AllUsers, removing a user requires that any GsObjectSecurityPolicies owned by the deleted user are moved to a new user, and the user

be removed from all groups. In addition, to ensure that work being done by the user being deleted is not lost, the SymbolLists of the deleted user are moved to a separate location where they can be periodically reviewed and manually dereferenced.

For more details on how this is handled, see “DeletedUserProfile and AllDeletedUsers” on page 152.

Remove User, with ObjectSecurityPolicies Going to the Current User

The following methods remove the user and reassign any GsObjectSecurityPolicies owned by the user to be removed to the UserProfile of the current user (the user executing this code, such as DataCurator).

```
AllUsers removeAndCleanup: aUserProfile.  
AllUsers removeAndCleanupUserWithId: 'aUserId' ifAbsent: aBlock
```

Remove User, with ObjectSecurityPolicies going to another User

The following methods remove the user and reassign any GsObjectSecurityPolicies owned by that user to another specific UserProfile.

```
AllUsers removeAndCleanup: aUserProfile  
    migrateSecurityPoliciesTo: anotherUserProfile.  
AllUsers removeAndCleanupUserWithId: 'aUserId'  
    migrateSecurityPoliciesToUserWithId: 'anotherUserId'  
    ifAnyAbsent: aBlock
```

For example,

```
topaz 1> printit  
AllUsers removeAndCleanup: (AllUsers userWithId: 'John')  
    migrateSecurityPoliciesTo: (AllUsers userWithId: 'Ann')  
System commitTransaction.
```

Users and Group membership

You can specify group memberships when creating users. Alternatively, you can add existing users to a UserProfileGroup, or add existing groups to a UserProfile. The methods to do this ensure that the set of groups that a UserProfile references matches the UserProfile references in the groups.

Adding a user to a group

```
UserProfileGroup >> addUser: aUserProfile  
    Add the given UserProfile to the receiver, and add the receiver to this UserProfile's  
    groups.  
UserProfile >> addGroup: 'groupNameString'  
    Add the UserProfileGroup with the given name to the receiver, and add the  
    receiver to the UserProfileGroup.  
UserProfile >> addToUserProfileGroup: aUserProfileGroup  
    Add the given UserProfileGroup to the receiver, and add the receiver to the  
    UserProfileGroup.
```

Removing a user from a group

`UserProfileGroup >> removeUser: aUserProfile`
 remove the given UserProfile from the receiver, and remove the receiver from this UserProfile's groups.

`UserProfile >> removeGroup: 'groupNameString'`
 Remove the UserProfileGroup with the given name from the receiver, and remove the receiver from the UserProfileGroup.

`UserProfile >> removeFromUserProfileGroup: aUserProfileGroup`
 Remove the given UserProfileGroup from the receiver, and remove the receiver from the UserProfileGroup.

Querying for Group members

`UserProfileGroup >> users`
 Return a set of UserProfiles that belong to the receiver.

`UserProfileGroup >> userIds`
 Return a set of UserIds for the UserProfiles that belong to the receiver.

Querying for a User's Groups

`UserProfile >> groups`
 Return the set of UserProfileGroups that this user belongs to.

`UserProfile >> groupNames`
 Return the group names for each UserProfileGroup that this user belongs to.

for example

```
! Create group
UserProfileGroup newGroupWithName: 'Sales'.
System commitTransaction.
! Add users
(UserProfileGroup groupWithName: 'Sales')
  addUser: (AllUsers userWithId: 'DataCurator');
  addUser: (AllUsers userWithId: 'GcUser').
! report the UserIds in that group
(UserProfileGroup groupWithName: 'Sales') userIds
%
anIdentitySet( 'DataCurator', 'GcUser')
```

8.4 Administering Users

List Existing Users

Privileges required: None.

There is no direct method within GemStone Smalltalk to list only the names of existing accounts. The following example shows one way to obtain that information:

```
topaz 1 > run
(AllUsers collect: [:each | each userId ]) asArray.
%
#1 SystemUser
#2 DataCurator
#3 Nameless
#4 SymbolUser
#5 GcUser
```

Modifying the UserId

Privileges required: ChangeUserId.

Updating the userId requires resetting the password for that user. The new user ID and password will take effect when you commit the current transaction. The names of special system users cannot be changed.

To modify the user ID of a GemStone user, execute the following expression:

```
(AllUsers userWithId: 'theUserId' )
  userId: 'newId'
  password: 'newPassword' .
```

An error is raised if *newId* is the userId of an existing UserProfile.

Modifying Password

Accounts that use external authentication (Unix, LDAP or SingleSignOn) do not manage the password within GemStone. This section applies only to accounts using GemStone authentication.

Users Changing Their Own Password

Privileges required: UserPassword, and the account must be using GemStone authentication.

In many cases, users set their own passwords and may be required to update them periodically. These users must be given the UserPassword privilege to do so, and use the method `UserProfile >> oldPassword:newPassword:` to update their password.

For example:

```
System myUserProfile
  oldPassword: 'oldPasswordString'
  newPassword: 'newPasswordString' .
```

Password choice is constrained by login security that is configured for the repository; see the discussion under “Limiting Choice of Passwords” on page 167.

The new password takes effect when you commit the current transaction.

A different method, requiring other privileges, is used by Administrators to update the password of another user.

Changing Another User's Password

Privileges required: OtherPassword, and the other user must be using GemStone authentication.

To modify the password of any GemStone user, execute the following expression.

```
(AllUsers userWithId: 'theUserId')
    password: 'newPasswordString' .
```

The new password takes effect when you commit the current transaction.

The password set by this method is not subject to the constraints described under “Limiting Choice of Passwords” on page 167, because this method can only be used by a user having the OtherPassword privilege. The password must not be the same as the UserId and must not be longer than 1024 characters.

Each password change of this type is noted in the GemStone security log, which currently is the Stone's log file. The entry includes the userId of the session making the change but not the new password.

Modifying defaultObjectSecurityPolicy

Each security policy maintains access authorization for its owner, the world, and an unlimited number of groups. There are three levels of authorization: none, read (read-only), and write (which includes read permission).

Determining Who Is Authorized to Read or Write in an Object Security Policy

Privileges required: read authorization for the security policy that controls access to this security policy, such as the DataCuratorObjectSecurityPolicy.

You can find out who is authorized to read or write objects in an security policy by sending it the message `asString`. For instance:

```
topaz 1> printit
PublishedObjectSecurityPolicy asString
%
anObjectSecurityPolicy, Number 6 in Repository SystemRepository,
Owner SystemUser write, Group Subscribers read, Group Publishers
write, World none
```

Changing the Authorization of an Object Security Policy

Privileges required: ObjectSecurityPolicyProtection or be the security policy's owner, and write authorization to the DataCuratorObjectSecurityPolicy.

The new authorization will take effect for logins following the commit of the current transaction.

CAUTION

Do not attempt to change the authorization of SystemObjectSecurityPolicy.

To change the authorization for a security policy, execute any (or all) of the following expressions.

```
theObjectSecurityPolicy ownerAuthorization: #authSym.
```

```
theObjectSecurityPolicy worldAuthorization: #authSym.
```

```
theObjectSecurityPolicy group: groupOrGroupString
authorization: #authSym.
```

NOTE

Exercise caution when changing the authorization for any security policy that a user may be using as his or her default or current security policy – whether or not the user owns the affected policy. If a user attempts to commit a transaction, but has created objects with a policy for which he or she no longer has write authorization, an error will be generated.

For example, to authorize the group Accounting to read (but not write) in user Eli's default security policy, you could execute the following expression:

```
(AllUsers userWithId: 'Eli') defaultObjectSecurityPolicy
  group: 'Accounting'
  authorization: #read.
```

If the group 'Accounting' does not exist, GemStone will return an error. See under "Create a group" on page 154 for information on creating a new group.

Remove a Group from an Object Security Policy's Authorization List

Privileges required: ObjectSecurityPolicyProtection, and write authorization for the security policy.

To remove a group from a security policy's list of authorized groups, execute an expression similar to the following:

```
theSecurityPolicy group: 'groupOrGroupString' authorization: #none
```

Change a User's Default Object Security Policy

Privileges required: DefaultObjectSecurityPolicy, and write authorization to the DataCurator ObjectSecurityPolicy.

Changes to another user's default security policy do not take effect until the next login.

To change a user's default security policy, execute the following expression:

```
(AllUsers userWithId: 'theUserId')
  defaultObjectSecurityPolicy: aNewSecurityPolicy
```

NOTE

If you change any user's default security policy (including your own) to a security policy for which that user lacks write authorization, and you subsequently commit the transaction, the affected user will no longer be able to log in to GemStone.

Modifying Privileges

Examining a User's Privileges

No privileges are required for this operation.

GemStone provides messages that allow you to determine which privileged methods a GemStone user may execute, and to change the privileges of any user. Naturally, you need the appropriate privileges to use those methods.

To find out which privileged methods a given user is permitted to execute, send the following message to the desired user's `UserProfile`:

```
(AllUsers userWithId: 'theUserId') privileges
```

This message returns an Array of Symbols. Table 8.1 on page 149 lists the Smalltalk operations controlled by each privilege.

Adding a Privilege

Privileges required: OtherPassword and write authorization to the `ObjectSecurityPolicy` of the user's `UserProfile`.

The new privileges will take effect when you commit the current transaction.

To add to a user's existing privileges, execute the following expression:

```
(AllUsers userWithId: 'theUserId') addPrivilege: aPrivilegeSym.
```

Here's an example that assigns three new privileges to user Bob:

```
topaz 1> printit
(AllUsers userWithId: 'Bob')
    addPrivilege: #SystemControl;
    addPrivilege: #SessionAccess;
    addPrivilege: #UserPassword .
System commitTransaction
%
```

Revoking a Privilege

Privileges required: OtherPassword and write authorization to the security policy of the user's `UserProfile`.

The privileges will be revoked when you commit the current transaction.

To revoke one (or more) of a user's existing privileges, execute the following expression:

```
(AllUsers userWithId: 'theUserId') deletePrivilege: aPrivilegeSym.
```

The following example revokes two of user Jane's privileges:

```
topaz 1> printit
(AllUsers userWithId: 'Jane')
    deletePrivilege: #SystemControl;
    deletePrivilege: #SessionAccess.
System commitTransaction
%
```

Reassigning All Privileges

Privileges required: OtherPassword and write authorization to the security policy of the user's UserProfile.

The new privileges will take effect when you commit the current transaction.

To redefine the full set of a user's privileges, perhaps adding some and revoking others, execute the following expression:

```
(AllUsers userWithId: theUserId) privileges: anArrayOfSym
```

This expression supersedes any previous privilege assignments. After the change is committed, only those privileges listed in the expression are valid for the user. Any privileges that were previously valid, but are not listed, are revoked.

For example:

```
topaz 1> printit
(AllUsers userWithId: 'Sam') privileges:
    #( UserPassword ) .
System commitTransaction
%
```

Modifying SymbolLists

Adding a SymbolDictionary to Your Own Symbol List

Privileges required: CodeModification.

You can add a dictionary to a symbol list by sending the message `UserProfile>>insertDictionary: aSymbolDictionary at: anIndex`. The change does not affect the transient copy of the symbol list that is used by another currently logged in session until that session commits or aborts.

This example inserts dictionary NewDict (which already exists in the Published dictionary) into the user's own symbol list:

```
topaz 1> printit
System myUserProfile
    insertDictionary: NewDict at: 2.
%
```

Inserting the new dictionary at index 2, as in the example, places it between the UserGlobals and the Globals dictionaries in the search order. Because symbol resolution depends on the order of dictionaries in a user's symbol list, the index used in this example may not be appropriate for all situations.

Adding a SymbolDictionary to Someone Else's Symbol List

Privileges required: OtherPassword and write permission to the security policy of the other user's SymbolDictionary.

This example inserts dictionary NewDict (which already exists in the Published dictionary) into user Jerry's symbol list:

```
topaz 1> printit
(AllUsers userWithId: 'Jerry')
    insertDictionary: NewDict at: 7.
System commitTransaction
%
```

Removing a SymbolDictionary from Your Own Symbol List

Privileges required: CodeModification.

You can remove a dictionary from a symbol list by sending the message `UserProfile>>removeDictionary: aSymbolDictionary`.

The change does not affect the transient copy of the symbol list that is used by another currently logged in session, until that session commits or aborts.

This example removes dictionary `OldDict` from the user's own symbol list:

```
topaz 1> printit
System myUserProfile
    removeDictionary: OldDict.
System commitTransaction
%
```

Removing a SymbolDictionary from Someone Else's Symbol List

Privileges required: OtherPassword and write permission to the security policy of the other user's `SymbolDictionary`,

This example removes dictionary `OldDict` from user Jerry's symbol list:

```
topaz 1> printit
(AllUsers userWithId: 'Jerry')
    removeDictionary: OldDict.
System commitTransaction
%
```

Disable and Enable User Logins

`UserProfiles` using GemStone authentication can be disabled in two ways; automatically, by violating password controls such as on the number of failed logins; and explicitly, by sending messages such as `disable`. An account that is disabled in GemStone cannot log in.

`UserProfiles` using external authentication can also be explicitly disabled in GemStone. However, they may also be unable to login due to violating the password controls enforced by the external authentication. GemStone does not detect, report, or re-enable accounts that are disabled by the external authentication mechanism. If the Unix or LDAP account cannot log in, or if Kerberos does not have a current ticket, logins will fail and you must correct the problems directly in Unix, LDAP, or Kerberos.

Once an account is disabled in GemStone, it must be re-enabled in GemStone before the account can log in again. For accounts using GemStone authentication, a new password must be assigned for that account by an administrator.

Explicitly Disable an Account in GemStone

Privileges required: OtherPassword. Logins cannot be disabled for special system accounts.

Users using any authentication scheme can be disabled using the methods `disable` or `disableWithReason:`. This prevents the user from logging in again, but does not affect currently logged in sessions.

For example:

```
topaz 1> printit
(AllUsers userWithId: 'Sam')
    disableWithReason: 'on Sabbatical'.
System commitTransaction
%
```

Re-enable an Account that is using GemStone authentication

Privileges required: OtherPassword.

To re-enable an account that uses GemStone authentication, an administrative user must login and reset the users's password. An account that becomes disabled, whether explicitly disabled or from security violations, will clear its password.

Users using GemStone authentication can be re-enabled using the methods `password:` or `reenableWithPassword:`.

For example, to reset Sam's password and require him to change his password the first time he logs in:

```
topaz 1> printit
(AllUsers userWithId: 'Sam')
    reenableWithPassword: 'AaBbCc';
    loginsAllowedBeforeExpiration: 1.
System commitTransaction
```

Re-enable an Account that is uses Unix, LDAP, or SingleSignOn authentication

Accounts using external authentication rely on the security procedures for the external authentication scheme. If the external authentication disables logins, the user will not be able to login using GemStone, but the account in GemStone is not disabled.

Accounts that were explicitly disabled in GemStone can be re-enabled using the method `reenable`. Since the password is managed outside GemStone, you do not need to supply a new password.

```
topaz 1> printit
(AllUsers userWithId: 'Mary') reenable.
System commitTransaction
```

Find Out Which Accounts Have Been Disabled in GemStone

Privileges required: OtherPassword.

The message `AllUsers findDisabledUsers` returns a `SortedCollection` of `UserProfiles` that were either disabled explicitly, or that use GemStone authentication and were disabled by a security precaution such as password expiration, login failure, etc. Accounts using external authentication and that have been disabled by the rules for that authentication are not disabled in GemStone, and will not be returned by this method.

For example:

```
topaz 1> level 1
topaz 1> printit
AllUsers findDisabledUsers
  collect: [:aUser | aUser userId ] .
%
an Array
  #1 Sam
  #2 Mary
```

See “Re-enable an Account that is using GemStone authentication” on page 164 for how DataCurator or another user with the OtherPassword privilege can reactivate an account.

Check If an Account Is Disabled

Privileges required: OtherPassword.

You can check if a particular account is disabled in GemStone by sending the message `isDisabled` to the account’s `UserProfile`.

For example:

```
topaz 1> printit
(AllUsers userWithId: 'Sam') isDisabled
%
true
```

Find Out Why an Account Was Disabled

Privileges required: OtherPassword.

You can find out why a particular account was disabled in GemStone by sending the message `reasonForDisabledAccount` to the account’s `UserProfile`.

For example:

```
topaz 1> printit
(AllUsers userWithId: 'Sam') reasonForDisabledAccount
%
LoginsWithSamePassword
```

If the account was disabled using the method `disableWithReason:`, this method will return that argument. Otherwise if the account was disabled by login security, it will return one of these Strings: 'PasswordAgeLimit', 'StaleAccount', 'LoginsWithSamePassword', or 'LoginsWithInvalidPassword'.

Disable and Enable Commits by User

Commits can be disabled for particular users to ensure “read only” access to the GemStone repository. These users can still log in and view data for which they have read or write authorization, and can modify objects, but they cannot commit and make any changes permanent.

This restriction is unrelated to authentication, and applies equally to all authentication schemes.

Disable Commits

Privileges required: OtherPassword. Commits cannot be disabled for special system users.

To disable commits for a user, execute the following expression:

```
(AllUsers userWithId: theUserId) disableCommits
```

This expression disables any commits for the given user, beginning with the next login of the user after the session making this change commits. If this user is currently logged in, it does not affect the user's session/s.

For example:

```
topaz 1> printit
(AllUsers userWithId: 'Sam') disableCommits.
System commitTransaction
%
```

Re-enable Commits for a User

Privileges required: OtherPassword.

To enable commits for a user, execute the following expression:

```
(AllUsers userWithId: theUserId) enableCommits
```

This expression enables commits for the given user, beginning with the next login after the session making this change commits.

Check If a User Can Commit

Privileges required: OtherPassword.

To test if commits have been disabled for a user account, execute the following expression:

```
(AllUsers userWithId: theUserId) isReadOnly
```

8.5 Configuring GemStone Authentication

GemStone authentication is the default authentication, using the UserId and password store with the UserProfile of the account. In older versions, this was the only way of authentication within GemStone, and must still always be used by the special system users – SystemUser, DataCurator, GcUser, SymbolUser, and Nameless.

GemStone authentication always requires a password to be set in the UserProfile. When you enable GemStone authentication, you must provide an initial password.

```
topaz 1> printit
(AllUsers userWithId: 'Mary')
  enableGemStoneAuthenticationWithPassword: 'AaBbCc'.
System commitTransaction.
%
```

Configuring GemStone Login Security

When authentication is done using #GemStone, there are a number of login security features available. You can:

- ▶ Constrain the choice of passwords to a certain pattern, ban particular passwords altogether, or ban reuse of a password by the same account

- ▶ Require users to change their passwords periodically (password aging)
- ▶ Limit the number of logins under a temporary password
- ▶ Disable accounts that have not logged in for a specified interval (account aging)
- ▶ Limit the number of concurrent sessions by a particular account
- ▶ Monitor failed login attempts and, if necessary, disable further login attempts on that account

In all cases, the password must not be the same as the UserId and must not be longer than 1024 characters.

Additional methods let you determine which accounts have been disabled by one of these security features and why a particular account was disabled.

CAUTION

GemStone records certain administrative changes to these security features in the Stone log. You may want to restrict access to that file.

The special system users - SystemUser, DataCurator, SymbolUser, GcUser, and Nameless - are never disabled by the security features.

Limiting Choice of Passwords

Privileges required: OtherPassword and write authorization to DataCuratorObjectSecurityPolicy.

You can constrain a user's choice of passwords in terms of pattern (such as the number of characters that repeat). Independently, you can establish a list of words that are disallowed as passwords, and you can keep a user from choosing the same password more than once.

The constraints described here apply only when a user changes his or her own password by using the message `UserProfile>>oldPassword:newPassword:` and only to password changes after the constraint is committed to the repository. That is, the constraints (other than the prohibition of `userId` as the password) do not apply to changing a user's password using the `password:` method (which requires OtherPassword privilege), and they do not invalidate existing passwords.

Table 8.3 lists the messages that you can use to set pattern constraints. You send these messages to the global object `AllUsers`. For example, to set the minimum password length to six characters, do this:

```
topaz 1> printit
AllUsers minPasswordSize: 6.
System commitTransaction
%
```

The default setting in all cases is 0, which means there is no constraint on the pattern.

Table 8.3 Ways to Constrain the Password Pattern

Message to AllUsers	Comments
<code>minPasswordSize:</code> <i>aPositiveInteger</i>	Sets the minimum number of characters in a new password; 0 means no constraint.
<code>maxPasswordSize:</code> <i>aPositiveInteger</i>	Sets the maximum number of characters in a new password; 0 disables the constraint. (The password String itself must not be longer than 1024 characters.)
<code>maxRepeatingChars:</code> <i>aPositiveInteger</i>	Sets the maximum number of adjacent characters that can have the same value; for example, 1 allows 'aba' but not 'aa'. 0 means no constraint.
<code>maxConsecutiveChars:</code> <i>aPositiveInteger</i>	Sets the maximum number of adjacent characters that can be an ascending or descending sequence, such as '123' or 'zyx' based on a case-sensitive comparison. 0 means no constraint.
<code>maxCharsOfSameType:</code> <i>aPositiveInteger</i>	Sets the maximum number of adjacent characters that can be of the same type (alpha, numeric, or special); for example, 3 allows 'abc4de' but not 'abcde'. 0 means no constraint.
<code>passwordRequiresUppercase:</code> <i>aBoolean</i>	Set the requirement that the password contain at least one uppercase character. false means no constraint.
<code>passwordRequiresLowercase:</code> <i>aBoolean</i>	Set the requirement that the password contain at least one lowercase character. false means no constraint.
<code>passwordRequiresSymbol:</code> <i>aBoolean</i>	Set the requirement that the password contain at least one character that is not alphanumeric. false means no constraint.
<code>passwordRequiresDigit:</code> <i>aBoolean</i>	Set the requirement that the password contain at least one digit. false means no constraint.

Any user can inquire about the current setting of a password pattern constraint by sending its corresponding Accessing message (that is, without the colon or argument shown in Table 8.3).

For example, to query for the current minimum size for a password:

```
topaz 1> printit
AllUsers minPasswordSize
%
6
```


Disallowing Particular Passwords

Privileges required: OtherPassword and write authorization to the DataCuratorObjectSecurityPolicy.

You can create a list of disallowed passwords by adding Strings to the AllUsers instance variable disallowedPasswords set. For instance:

```
(AllUsers disallowedPasswords)
  addAll: #( 'Mother' 'apple_pie' )
```

The default is an empty set.

Additions to this list affect only new passwords requested after the additions are committed; that is, additions do not invalidate existing passwords. If a user attempts to change that account's password to one of the Strings in disallowedPasswords, a SecurityError is signaled.

Any user can examine the current list of globally disallowed passwords by sending the message AllUsers disallowedPasswords.

Disallowing Reuse of Passwords

Privileges required: OtherPassword and write authorization to DataCuratorObjectSecurityPolicy.

You can prevent each user from choosing the same password more than once by setting AllUsers disallowUsedPasswords to true. By default, disallowUsedPasswords is false.

When reuse of passwords is disallowed, GemStone maintains a separate encrypted set of old passwords for each user. Each time a user invokes oldPassword:newPassword:, the new password is checked against the prior passwords for that account. If the new password matches a prior one, a SecurityError is signaled.

Disallow All Previously Used Passwords

To disallow password reuse, use the method:

```
AllUsers disallowUsedPasswords: aBoolean.
```

Disallow a Specific Number of Previous Passwords

Privileges required: OtherPassword and write authorization to DataCuratorObjectSecurityPolicy.

To disallow a fixed number of previously-used password, but allow earlier passwords, use the following:

```
AllUsers numberOfDisallowedPasswords: anInteger
```

anInteger must be a number between 0 and 65535; 0 means that the user may not reuse any previously-used passwords. The limit on the number of disallowed passwords has no effect if disallowUsedPasswords is false.

For example:

```
AllUsers disallowUsedPasswords: true.
AllUsers numberOfDisallowedPasswords: 10.
```

Clearing a User's Disallowed Old Passwords

Privileges required: OtherPassword.

You can clear the set of old passwords so that they can be reused. As mentioned above, this set is maintained for each user when the AllUsers instance variable disallowUsedPasswords is set to true.

The following example clears the remembered passwords for Mary's account:

```
(AllUsers userWithId: 'Mary')
  clearOldPasswords.
```

Password Aging – Require Periodic Password Changes

You can configure your system so users must change their password periodically. This can be configured at the repository-wide level, or for individual users. Note that if you are not using GemStone login authorization, password aging does not apply.

Privileges required: OtherPassword and write authorization to DataCuratorObjectSecurityPolicy. The special GemStone accounts are not disabled by password aging.

Repository-Wide Password Aging

You can require users to change their password periodically by setting up a password age limit. This can be set for all users in the repository, and can be overridden for specific individual users.

To set a password page limit for all users in the repository, for example to 120 days:

```
AllUsers passwordAgeLimit: 120 * 24 .
```

The passwordAgeLimit is added to the time the password was last changed to determine when the password will expire. A setting of 0 (the default) disables password aging.

Each time this method is invoked, the action is recorded in the Stone's log.

If a user does not change the account's password within the specified interval, the account is disabled, and attempts to log in result in an error.

DataCurator or another user with the OtherPassword privilege can reactivate the disabled account by giving it a new password; see "Re-enable an Account that is using GemStone authentication" on page 164 for details.

Password Age Limits for Individual Users

To override the repository-wide setting for password aging, you can set a password age limit for a specific user. For example, to set the limit for a Mary to 5 days:

```
(AllUsers userWithId: 'Mary')
  passwordAgeLimit: 5 * 24.
```

If repository-wide password aging is enabled, you can also override this for individual users, such as managers or administrators, either by setting the password page limit to 0, or setting the password to never expire. For example, to prevent the password used by batch jobs from expiring, execute:

```
(AllUsers userWithId: 'BatchUser')
  setPasswordNeverExpires: true.
```

Repository-Wide Password Expiration Warning

Privileges required: OtherPassword and write authorization to DataCuratorObjectSecurityPolicy. This does not apply to system users, and has no effect if password aging is not enabled.

You can provide an automatic warning to users repository-wide whose password is about to expire. For example, to warn users who log in within five days of the time their password will expire, do this:

```
AllUsers passwordAgeWarning: 5 * 24.
```

Logins within *numberOfHours* prior to expiration cause a SecurityError to be signaled.

Per-User Password Expiration Warning

You can provide a similar automatic warning to a specific user. For example, to warn Mary within three days of the time her password will expire, do this:

```
(AllUsers userWithId: 'Mary') passwordAgeWarning: 3 * 24.
```

Finding Accounts with Password About to Expire

Privileges required: OtherPassword.

You can find out which accounts have a password within the warning period set by `passwordAgeWarning`. To do this, send the message `findProfilesWithAgingPassword` to `AllUsers`. For example:

```
topaz 1> printit
AllUsers findProfilesWithAgingPassword
  collect: [ :u | u userId] .
%
an OrderedCollection
#1 qa1
#2 qa2
#3 qa3
```

Finding Out When a Password Was Changed

Privileges required: OtherPassword.

You can find out the last time the password was changed for a particular `userId` by sending the message `lastPasswordChange` to that account's `UserProfile`. This example converts the `DateTime` returned to a particular pattern based on MM/DD/YY:

```
topaz 1> printit
(AllUsers userWithId: 'qa2') lastPasswordChange US12HrFormat
%
04/06/17 11:28 AM
```

Account Aging – Disable Inactive Accounts

You can configure your system so users must log in periodically, by disabling accounts for which there has been no login for a specified length of time. This can be configured at the repository-wide level, or for individual users.

Privileges required: OtherPassword and write authorization to DataCuratorObjectSecurityPolicy. The special GemStone accounts are not disabled by stale account aging.

Repository-Wide Stale Account Aging

To do this, send the message `staleAccountAgeLimit: numberOfHours` to `AllUsers`. This example disables accounts when they have not logged in for 30 days:

```
AllUsers staleAccountAgeLimit: 30 * 24.
```

Each time this method is invoked, the action is recorded in the Stone log.

A setting of 0 (the default) disables account aging.

`DataCurator` or another user with the `OtherPassword` privilege can reactivate the disabled account by giving it a new password; see “Re-enable an Account that is using GemStone authentication” on page 164 for details.

Per-User Stale Account Aging

You can override the repository-wide setting for account aging for a specific user using `UserProfile>>staleAccountAgeLimit: numberOfHours`.

For example, for the ‘Auditor’ account who may log in less frequently, you can set up a 180-day stale account age limit. This will override a repository-wide 30 day setting.

```
(AllUsers userWithId: 'Auditor')
  staleAccountAgeLimit: 180 * 24.
```

Finding Out When an Account Last Logged In

Privileges required: `OtherPassword`.

If at least one age limit applies to an account, you find out when that account last logged in by sending the message `lastLoginTime` to that account’s `UserProfile`. For example:

```
topaz 1> printit
(AllUsers userWithId: 'Mary') lastLoginTime US12HrFormat
%
04/06/17 01:40 PM
```

The time of the last login is maintained only if `loginsAllowedBeforeExpiration` is set in that `UserProfile` or if at least one of these instance variables is set in `AllUsers` or the specific `UserProfile`: `passwordAgeLimit`, `passwordAgeWarning`, or `staleAccountAgeLimit`. If none of these features are enabled, the `lastLoginTime` may be nil, the time of the account creation, or a time representing a login during an earlier period when one of these features was enabled. This is also true if a feature that enables `lastLoginTime` recording has been enabled on more recently than the last login of the user.

The data curator may explicitly set the time of the last login, using the method `UserProfile >> lastLoginTime:.`

Enabling Account Aging and lastLoginTime

Privileges required: `OtherPassword`.

The time of the last login is recorded for a `UserProfile` only if `loginsAllowedBeforeExpiration` is set in that `UserProfile` or if at least one of these instance variables is set in `AllUsers` or the specific `UserProfile`: `passwordAgeLimit`, `passwordAgeWarning`, or `staleAccountAgeLimit`. This is to avoid the commit during login, which is required to record the `lastLoginTime`.

Use caution in enabling account aging on existing repositories. Enabling account aging may result in user accounts being disabled, if there happens to be a `lastLoginDate` recorded in the `UserProfile`. This may be due to a login date recorded in an earlier period due to

changes in login security in your application, or UserProfiles in an repository progressively upgraded from historical versions of GemStone that routinely set the lastLoginDate.

To avoid this issue, when you enable account aging, you can set the lastLoginTime to the current date, or to nil, for all affected UserProfiles. A nil setting disables account aging checks, allowing the aging period to being with the next login.

```
topaz 1> printit
AllUsers passwordAgeLimit: 120 * 24.
AllUsers do: [:aUserProfile |
    aUserProfile lastLoginTime: DateTime now.
].
System commitTransaction.
%
```

Limit Logins Until Password Is Changed

Privileges required: OtherPassword.

When you assign a password to an account, you can make the password temporary by limiting the number of times it can be used. This limitation applies only to a specific account, that is, to the UserProfile that is the receiver of the message. It is intended for use with a new or reactivated account as a means of ensuring that the user changes the password. For example, the following limits the account “Mary” to two more logins under the current password:

```
(AllUsers userWithId: 'Mary')
    loginsAllowedBeforeExpiration: 2.
```

A setting of 0 (the default) disables this feature.

The limit remains in effect until the user changes the password. Once the password is changed, the limit for that account is set to 0. The password will not expire again unless a new limit is set by repeating `loginsAllowedBeforeExpiration:`.

If the limit is exceeded before the password is changed, the system disables the account. DataCurator or another user with the OtherPassword privilege can reactivate the disabled account by giving it a new password; see “Re-enable an Account that is using GemStone authentication” on page 164 for details.

The special user accounts are not disabled by this mechanism.

Limit Concurrent Sessions by a Particular UserId

Privileges required: OtherPassword.

You can limit the number of concurrent sessions logged in under a particular userId. For example, the following limits the userId “qa2” to four concurrent sessions:

```
(AllUsers userWithId: 'qa2')
    activeUserIdLimit: 4.
```

A setting of 0 (the default) disables this feature.

If a user attempts to log in when the maximum number of sessions for that userId are already logged in, the login is denied and a SecurityError is signalled.

Limit Login Failures

Record Login Failures

The Stone repository monitor keeps track of login failures (incorrect passwords) for each account and can write that information to the Stone's log.

By default, messages are logged when the same account fails login attempts 10 or more times within ten minutes. You can change the default limits by setting the `STN_LOG_LOGIN_FAILURE_LIMIT` (page 321) and `STN_LOG_LOGIN_FAILURE_TIME_LIMIT` (page 321) configuration options.

The log message gives the following information:

```
---Mon 1 May 2017 09:39:40 PDT ---
    GemStone user Mary has failed on 10 attempt(s)
        to log in within 1 minute(s).
    The last attempt was from user account writer1 on hostname
    docs.
```

Disabling Further Login Attempts

If login failures continue, the Stone repository monitor can disable the account. By default, the account is disabled when the number of failures exceeds 15 within 15 minutes. You can change the default limits by setting the `STN_DISABLE_LOGIN_FAILURE_LIMIT` (page 315) and `STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT` (page 315) configuration options.

Subsequent attempts to login as that account result in the following error message:

```
Login failed:  the GemStone userId/password combination is
invalid or expired.
```

The special user accounts are not disabled by login failures.

DataCurator or another user with the OtherPassword privilege can reactivate the disabled account by giving it a new password; see "Re-enable an Account that is using GemStone authentication" on page 164 for details.

8.6 Configuring UNIX Authentication

When UNIX account authentication is enabled for a user, they will enter their UNIX account password instead of their GemStone password. Password management for this user then becomes the Unix password management; sending messages to change the GemStone password are not useful, and these accounts are not subject to GemStone's password restriction, aging, and failed login mechanisms.

UNIX authentication uses PAM (pluggable authentication module) , and PAM must be configured on the system in order to use UNIX Authentication. Login will look for a module `gemstone.gem`. See the *Installation Guide* for your server platform for details.

The GemStone `userId` may be the same as the UNIX `userId`, or they may be different. When you enable UNIX authentication for an account, you may specify the UNIX `userId` associated with the GemStone `UserProfile` and this will be used for authentication. Using `nil` means to use the GemStone `userId` as the UNIX `userId`.

```
(AllUsers userWithId: GemStoneUserId)
    enableUnixAuthenticationWithAlias: UNIXUserIdOrNil
```

for example, if Mary's UNIX `userId` is `msmith`, you can use the first form if her GemStone `userId` is `Mary`. If her GemStone `userId` is also `msmith` you leave the argument `nil`.

```
(AllUsers userWithId: 'Mary')
  enableUnixAuthenticationWithAlias: 'msmith'.
```

```
(AllUsers userWithId: 'msmith')
  enableUnixAuthenticationWithAlias: nil.
```

To verify that the UNIX `userId` exists, execute:

```
System unixUserIdExists: UNIXUserId
```

8.7 Configuring LDAP Authentication

An LDAP (Lightweight Directory Access Protocol) server consolidates and centralizes user authentication, and can be used to authenticate logins to the GemStone server. UNIX authentication, which uses PAM, may ultimately use LDAP if PAM is configured to use LDAP. When you configure GemStone to use LDAP authentication, it goes to LDAP directly, rather than using PAM.

To use LDAP for GemStone authentication, you must have an LDAP server available. When a `UserProfile` has been configured for LDAP authentication, on login, GemStone performs an LDAP bind to authenticate the `userId`.

In most cases, the LDAP bind requires a Distinguished Name (DN), which is the unique identifier for an entry. The DN includes both the `userId` and domain information, e.g. `'uid=msmith,ou=employees,dc=somecompany,dc=com'`. GemStone composes the DN based on the arguments provided when you configure LDAP authentication.

To configure a user to use LDAP for authentication, use the following method:

```
(AllUsers userWithId: GemStoneUserId)
  enableLDAPAuthenticationWithAlias: LDAPUserIdOrNil
  baseDn: baseDn
  filterDn: filterDn
```

Userid or Alias

As with UNIX authentication, if the LDAP `userId` is the same as the GemStone Id, you may pass in `nil` for the argument `LDAPUserIdOrNil`; otherwise, pass in the LDAP `userId`. The user will use their GemStone `userId` and the password for their LDAP account to log in.

Fully Qualified DN

You can configure the LDAP authentication for a particular user with the specific DN, with `'%s'` replacing the `userId`, for the `baseDn`: argument. In this case you should pass in `nil` for the `filterDn`: argument, which will disable the query.

For example, to configure authentication to use a specific fully qualified DN:

```
topaz 1> printit
(AllUsers userWithId: 'Mary')
  enableLDAPAuthenticationWithAlias: 'msmith'
  baseDn: 'uid=%s,ou=employees,dc=somecompany,dc=com'
  filterDn: nil.
System commitTransaction.
%
```

Search for DN

You can also configure authentication to include the base domain information in the `baseDn:` argument, and include a filter in the `filterDn:` argument. The filter must contain `'%s'` in the position for the `userId`. GemStone will perform a query to get the full DN given the base and filter information.

The base and filter information is provided at the time the authentication is configured, not at login time, so a search option is particularly useful if the LDAP structure is likely to be modified.

To configure authentication to do a search for the given user:

```
topaz 1> printit
(AllUsers userWithId: 'Mary')
  enableLDAPAuthenticationWithAlias: 'msmith'
  baseDn: 'dc=somecompany,dc=com'
  filterDn: '(uid=%s)'.
System commitTransaction.
%
```

LDAP authentication without anonymous binds

By default, GemStone login authentication requires that the LDAP server allow anonymous binds. For installations that disallow anonymous binds, you may configure instances of `LdapDirectoryServer` with the information required for authenticated binds.

`AllLdapDirectoryServers` is a global collection of instance of `LdapDirectoryServer`. Each instance of `LdapDirectoryServer` contains authentication information for an particular LDAP server.

During login, If there are existing instances of `LdapDirectoryServer` in `AllLdapDirectoryServers`, then these are used, in order; normally this would include a primary LDAP server and an alternate. If and only if the bind to the first LDAP server fails, the second is tried, and so on. If no binds are successful, then the login will fail.

If `AllLdapDirectoryServers` is empty, then login will use anonymous bind.

To create an `LdapDirectoryServer`, and add it to the global list, use the following method:

```
LdapDirectoryServer class >> newWithUri: uri bindDN: aBindDn
  password: password
```

If LDAP authentication is setup, using

```
enableLDAPAuthenticationWithAlias: aString baseDn: baseDn
  filterDn: filterDn
```

then either anonymous or authenticated binds are performed to authenticate the user password.

For example, to set up a user to use LDAP authentication, and setup a single LDAP server to authenticate binds, the following code would be executed.

```
aUserProfile
  enableLDAPAuthenticationWithAlias: nil
  baseDn: 'ou=Users,dc=gemtalksystems,dc=com'
  filterDn: '(uid=%s)' .

LdapDirectoryServer
  newWithUri: 'ldaps://ldap.gemtalksystems.com'
  bindDN: 'uid=bindUser,ou=Users,dc=gemtalksystems,dc=com'
  password: 'swordfish' .
```

Users are individually configured to use LDAP authentication, but LDAP directories are configured for the repository as a whole, so this step is only done once. All users use the same bind authentication, though they use their own individual passwords to perform the actual account authentication.

Note that the instances of `LdapDirectoryServer` store the encrypted bind password. This is not the actual user password to login to GemStone; this bind password allows you to perform the user password authentication.

To determine if a particular `LdapDirectoryServer` exists, use the following method, which returns either the `LdapDirectoryServer` or `nil` if it is not found:

```
LdapDirectoryServer >>findServerWithUri: aUriString
```

to remove an `LdapDirectoryServer`, use the method:

```
LdapDirectoryServer >> removeServerWithUri: aUriString
```

Validating passwords

Once logged in, you can perform an validation call to the LDAP server that passes in the LDAP authentication as well as the user authentication, using this method:

```
System class >> validatePasswordUsingLdapServers:baseDn:
  filterDn:userId:password:bindDn:bindPassword:
```

The additional bind arguments should be `nil` if authenticating in explicit mode, or with anonymous bind. For example:

Explicit mode

```
System validatePasswordUsingLdapServers:
  (Array with: 'ldaps://myldap.mydomain.com')
  baseDn: 'uid=%s,ou=Users,dc=mycompany,dc=com'
  filterDn: nil
  userId: 'MyUserId' password: 'swordfish'
  bindDn: nil bindPassword: nil
```

Search mode with anonymous bind

```
System validatePasswordUsingLdapServers:
  (Array with: 'ldaps://myldap.mydomain.com')
  baseDn: 'ou=Users,dc=mycompany,dc=com'
  filterDn: '(uid=%s)'
  userId: 'MyUserId' password: 'swordfish'
  bindDn: nil bindPassword: nil
```

Search mode with authenticated bind

```
System validatePasswordUsingLdapServers:
    (Array with: 'ldaps://myldap.mydomain.com')
    baseDn: 'ou=Users,dc=mycompany,dc=com'
    filterDn: '(uid=%s)'
    userId: 'MyUserId' password: 'swordfish'
    bindDn: 'LdapBindUser' bindPassword: 'LdapBindPassword'
```

For further details, see the method comments in the image.

8.8 Configure SingleSignOn Authentication

If your system is setup with Kerberos, you can configure GemStone authentication to use SingleSignOn, which checks for a current Kerberos ticket and avoids the requirement that a user should provide a password on login. Using SingleSignOn (SSO), the “User Password” field in the login parameters is left empty.

When Kerberos is not running or has no current ticket, the user cannot login. For this reason, SystemUser may not be setup to use SSO. SystemUser always requires GemStone authentication.

Kerberos concepts

For details on how to install or configure Kerberos at your site, consult your System Administrators. GemStone uses Kerberos v5 Release 1.15 with AES encryption.

Note that on Windows, Kerberos is built in as part of SSPI, and SingleSignOn can only be used on Windows clients that are part of a Windows Domain, either Samba or an MS Windows server.

Kerberos uses the term "realm" for a domain within an installation/organization. Realms are generally the DNS domain in upper case; for example, at GemTalk the realm would be GEMTALKSYSTEMS.COM.

The term "principal" is used for any unique identity, including UNIX user ids, hosts, and services.

User principles are identified as name@REALM, e.g.

```
maryb@BIGCORPORATION.COM
```

When a user authenticates using Kerberos (e.g by UNIX login or by using kinit), the user enters their password, and Kerberos provides a ticket – more specifically, a **ticket granting ticket (TGT)**. This is cached on the user’s host, and is used to create specific tickets for requested services at the time they are requested. The TGT is usually put under `/tmp`, and is valid for a limited period, by default usually 10 hours.

In addition to Kerberos user principals, you will need to create a Kerberos **service principal** for GemStone. The service principal is of the form `Service/fully.qualified.domain.name@REALM`. The name of the service principal for authentication in GemStone is GEMSTONE64. So, for example, a service principal might be

```
GEMSTONE64/nodename.bigcorporation.com@BIGCORPORATION.COM
```

You will need to create a **keytab file** with the GEMSTONE64 service principals for each individual host node. This keytab file will be used for authorization; any user who has

authenticated via Kerberos and has access to the keytab file, and has their GemStone user account setup for SingleSignOn, will be able to do a passwordless login to GemStone.

KerberosPrincipal and AllKerberosPrincipals

Instances of the class `KerberosPrincipal` define the association between the name of a Kerberos principal and GemStone login information. An instance of `KerberosPrincipal` has the following information:

<code>name</code>	(Symbol) the name for the kerberos principal; must be unique.
<code>loginUserProfile</code>	a <code>UserProfile</code> , or <code>nil</code> for a shared <code>KerberosPrincipal</code> .
<code>loginUserProfileGroups</code>	an <code>IdentitySet</code> of <code>UserProfileGroups</code> , to support a shared <code>KerberosPrincipal</code> , or <code>nil</code> .
<code>loginAsAnyoneEnabled</code>	a Boolean; true if any <code>UserProfile</code> can use this <code>KerberosPrincipal</code> .

There are a number of ways that a repository can be configured to allow one or more `UserProfiles` to login using one or more `KerberosPrincipals`.

- ▶ There may be a one-to-one relationship between a `UserProfile` and a `KerberosPrincipal`, with each referencing the other. In this case `loginUserProfileGroups` and `loginAsAnyoneEnabled` are usually `nil`.
- ▶ One or more `UserProfile/s` may reference a `KerberosPrincipal` with `loginUserProfileGroups` configured as a set of `UserProfileGroups`. The `KerberosPrincipal` has a `nil` `loginUserProfile` and `loginAsAnyoneEnabled` set to `false`. Any user that is a member of any of these groups can use this `KerberosPrincipal` to log in.
- ▶ One or more `UserProfile/s` may reference a `KerberosPrincipal` that has a `nil` `loginUserProfile`, and with `loginAsAnyoneEnabled` set to `true`. This allows any user to configure `SingleSignOn` using that `KerberosPrincipal`.

`KerberosPrincipals` are created using class protocol in `KerberosPrincipal`, and are automatically added to the global collection `AllKerberosPrincipals`. You can look up a `KerberosPrincipal` by name. The `AllKerberosPrincipals` global should not be accessed directly.

Setting up Kerberos Authentication in GemStone

There are several steps to setting up `SingleSignOn` for a user account. You must:

- ▶ create the `KerberosPrincipal` specific to that user
- ▶ use that `KerberosPrincipal` as an argument to enable `#SingleSignOn` authentication
- ▶ configure the environment for the user that will login with the keytab file location.

1. Create an instance of KerberosPrincipal for the User

A UserProfile who will use SSO is associated with a specific instance of KerberosPrincipal. For example, to create a Kerberos user principle for the GemStone user named Mary, use an expression such as this:

```
KerberosPrincipal
    newPrincipalWithName: 'maryb@BIGCORPORATION.COM'
    loginUserProfile: (AllUsers userWithId: 'Mary').
```

This creates the KerberosPrincipal and adds it to AllKerberosPrincipals.

2. Enable SingleSignOn for that UserProfile using the new KerberosPrincipal

Use the Kerberos Principle when enabling SingleSignOn, using UserProfile >> enableSingleSignOnAuthenticationWithPrincipal:.

3. Configure the gem's environment to set the keytab file location

The Gem configuration parameter GEM_KERBEROS_KEYTAB_FILE must be set in order to specify the location of the keytab file.

For example, enter this in the configuration file that will be used by Gem sessions. This may be the default \$GEMSTONE/data/system.conf, or another configuration file, as described in "How GemStone Uses Configuration Files" on page 285.

```
GEM_KERBEROS_KEYTAB_FILE =
    '/export/localnew/common/kerberos/gemtalk.keytab';
```

4. Verify login without password

Verify authentication; make sure the user account for maryb is authenticated using Kerberos, enter the GemStone user name (Mary) in the login parameters, and leave the password empty.

Example 8.1 Setting up SingleSignOn for a single account

```
| prin user |
user := AllUsers userWithId: 'Mary'.
prin := KerberosPrincipal
    newPrincipalWithName: 'maryb@GEMTALKSYSTEMS.COM'
    loginUserProfile: user.
user enableSingleSignOnAuthenticationWithPrincipal: prin.
System commitTransaction.
```

Using Groups to authenticate with Kerberos

A KerberosPrincipal often will correspond to a specific GemStone UserProfile. However, you may use groups to configure your system so multiple GemStone UserProfiles can login using the same KerberosPrincipal.

To do this, create a KerberosPrincipal with a nil loginUserProfile. Create a new group and make the users part of the group, and add the group to the KerberosPrincipal. You can then use this KerberosPrincipal to enable SignSignOn for all the users in the group

For example, the following code uses jsmith's principal to login as both 'john' and 'mary'.

Example 8.2 Setting up SingleSignOn using Groups

```
| prin user1 user2 group |
user1 := AllUsers userWithId: 'john'.
user2 := AllUsers userWithId: 'mary'.
group := UserProfileGroup
        newGroupWithName: 'KerberosAdminLogin'.
user1 addGroup: group groupName.
user2 addGroup: group groupName.
prin := KerberosPrincipal newPrincipalWithName:
        'jsmith@GEMTALKSYSTEMS.COM' loginUserProfile: nil.
prin addGroup: group.
group users do: [:aUserProfile | aUserProfile
        enableSingleSignOnAuthenticationWithPrincipal: prin].
System commitTransaction.
```

KerberosPrincipal available to all users

You can configure all users on your system to share the `KerberosPrincipal` by using

```
aKerberosPrincipal loginAsAnyoneEnabled: true
```

This is equivalent to creating a group that includes all users and using this for the `KerberosPrincipal`'s group.

Then, any user (except `SystemUser`) can set their authentication to `#SingleSignOn` using the principal *aKerberosPrincipal*.

Example 8.3 Sharing a KerberosPrincipal

```
| prin user1 user2 |
user1 := AllUsers userWithId: 'Mary'.
user2 := AllUsers userWithId: 'John'.
prin := KerberosPrincipal newPrincipalWithName:
        'jsmith@GEMTALKSYSTEMS.COM' loginUserProfile: nil.
prin loginAsAnyoneEnabled: true.
user1 enableSingleSignOnAuthenticationWithPrincipal: prin.
user2 enableSingleSignOnAuthenticationWithPrincipal: prin.
System commitTransaction.
```

8.9 Tracking User Logins

GemStone provides to options related to user logins: logging of each user login and logout, and a hook that allows you to execute specific code when a specific user logs in.

These features apply for all authentication schemes.

Login logging

It is sometimes useful to keep a log recording when each session logs in and out. You can configure your system to record login/logout events for each session in the repository. Other related operations, such as Stone startup and shutdown, are also recorded.

Tracking login/logout events is disabled by default. It is enabled by setting the `STN_LOGIN_LOG_ENABLED` configuration option to `True` in the configuration file used by the Stone, prior to Stone startup.

Once this is enabled for the repository, by default, all sessions that login to the stone will have logins and logouts logged.

Specific `UserProfiles` can be configured to not log events on login and logout; this can help avoid having the log cluttered with system logins. For example, you may not want to track when the `GcUser` logs in and out. This is done with the method `UserProfile >> disableLoginLogging`. After this is executed, the particular `UserProfile` will not have logins or logouts recorded in the log file.

Logins and logouts are recorded to a text file named `stoneName_login.log`, in the same directory as the Stone log. Each log entry is on an individual line, with the following fields:

```
TimestampString TimeStampSeconds EventKind UserName SessionId
ProcessId RealUserID EffectiveUserID HostName GemIPAddress
ClientIPAddress NumCommits LoginUserId KerberosPrincipal
```

For example:

```
"02/28/17 10:49:13.404 PST" 1488307753 STARTUP Stone 0 2045 631 631
kata.gemtalksystems.com ::ffff:204.45.141.15 0.0.0.0 0 631 ''
"02/28/17 10:51:04.111 PST" 1488307864 LOGIN DataCurator 5 2641 631
631 kata.gemtalksystems.com ::ffff:204.45.141.15
::ffff:10.94.141.15 0 531 ''
"02/28/17 10:51:05.792 PST" 1488307865 SHUTDOWN Stone 0 2045 631 631
kata.gemtalksystems.com ::ffff:204.45.141.15 0.0.0.0 0 631 ''
```

Login Hook

The `loginHook` is an optional feature that allows you to specify code to be executed each time a specific `userProfile` logs in.

The argument to the method `UserProfile >> loginHook`: specifies either a symbol, which must be a unary selector of an instance method that was added to `UserProfile`, or a zero-argument block. When the `userProfile` logs in, if the `loginHook`: is not nil, than the method associated with the selector, or the block, is executed.

For example:

```
topaz 1> printit
(AllUsers userWithId: 'Mary')
  loginHook: [MyApplicationLogger logLogin: 'Mary']
System commitTransaction.
%
```

Debugging

The loginHook code will be executed each time that user logs in. If there are errors in the loginHook code, the login does not complete and the user will not be able to log in.

Logging in as another user with otherPassword privilege may be required, to reset the loginHook code and allow the user to login.

SystemUser can enable to configuration parameter STN_ALLOW_NO_SESSION_INIT using the runtime parameter #StnAllowNoSessionInit. When this is set, the login reports the error but will complete, which may allow you to debug the loginHook code.

Managing Repository Space

The *repository* is the logical unit that represents the universe of shared objects that are stored within a GemStone/S 64 Bit system. The logical repository maps to one or more physical *extent* files in the file system, or to data on one or more raw disk partitions.

This chapter explains how the repository grows, and tells you how to perform a number of administrative tasks related to the repository and extents.

The Repository and Extents (page 185)

describes the relationship between the repository and the extents, and why the repository grows.

Adding and Removing Extents (page 187)

provides instructions on adding and removing extents from the repository.

Reallocating Existing Objects Among Extents (page 190)

describes how to change the way objects are distributed over the extents.

Shrinking the Repository (page 191)

provides instructions for reducing the size of the repository extents.

Checking Page Fragmentation (page 194)

describes page fragmentation and how to address it.

Disk Space and Commit Record Backlogs (page 194)

avoiding running out of space due to an idle session creating a commit record backlog.

Recovering from Disk-Full Conditions (page 196)

describes the actions taken by the Stone to prevent running out of space, what to do to avoid this condition, and how to recover.

9.1 The Repository and Extents

Within GemStone Smalltalk, the repository – the logical unit holding the universe of shared GemStone objects – is the single instance of Class Repository, with the name `SystemRepository`. The logical repository is represented on disk by one or more

physical *extent* files in the file system raw disk partitions. This is described under “Extents, Tranlogs, and disk space” on page 32.

Whenever GemStone performs a *checkpoint*, it makes sure that transactions committed before the checkpoint have been written to the repository extents. The maximum time between checkpoints is set by the `STN_CHECKPOINT_INTERVAL` configuration option; the default is five minutes, but various factors may cause a checkpoint to occur sooner. The checkpoint limits the amount of time that is needed to recover from a system crash by guaranteeing that the data for the transaction is written to the extent and not just to the transaction log. For information, see “Controlling Checkpoint Frequency” on page 58.

Repository Growth

Repository extents not only have to hold the data in your database, they also need to hold the changes all the users make, and coordinate the views of each user so the user has a consistent view of the data. All these activities require space in the extents.

The repository begins in the compact form of `$GEMSTONE/data/extent0.dbf`. As repository activity progresses, the extent file expands for a variety of reasons, in increments of 16 MB. Not only does your new application data require space, but space is required for internal structures that organize and manage the objects. Sessions, and their transactional views of the repository, also require space.

Garbage objects – objects that are no longer referenced, or the older versions of objects – also use space in the repository until they are garbage collected. To manage the size of the extents, you need to regularly perform garbage collection on your GemStone repository. The frequency can vary from monthly to daily, depending on the amount of activity in the repository. Without garbage collection, the repository will continue to grow and be filled with wasted space. See Chapter 14, “Managing Growth”, for a discussion of garbage collection in GemStone.

How To Check Free Space

Use the methods `Repository>>fileSize` and `Repository>>freeSpace` to obtain reports about the logical repository as a whole.

The result of the message `fileSize` is the total size of the repository in bytes, including all extent files. If the repository consists of a single extent file, it is ordinarily the same result as you would obtain by using the operating system commands to find file size. For example:

```
topaz 1> printit
SystemRepository fileSize
%
153092096
```

The result of `freeSpace` tells how much space (in bytes) is available for allocation within the repository at its current size. Free space is equal to the sum (for all extents in the repository) of the number of free pages in each extent multiplied by the page size. This space does not include fragments on partially filled data pages.

```
topaz 1> printit
SystemRepository freeSpace
%
26411008
```

Depending on the configuration options selected and the available disk space, the Stone repository monitor may be able to create additional free space by enlarging the repository. If your configuration has more than one extent, use `Repository>>fileSizeReport` to generate statistics about each individual extent and also totals for the entire repository. (The heading “Extent #1” identifies the primary extent regardless of its file name, which initially is `extent0.dbf`.)

Example 9.1 Checking free space using `fileSizeReport`

```
topaz 1> printit
SystemRepository fileSizeReport
%
Extent #1
  Filename = /users/extents/extent0.dbf
  File size =      208.00 Megabytes
  Space available = 3.69 Megabytes
-----
Extent #2
  Filename = /users/extents/extent1.dbf
  File size =      74.00 Megabytes
  Space available = 4.77 Megabytes
-----
Totals
  Repository size = 282.00 Megabytes
  Free Space =      8.45 Megabytes
```

The amount of free space in the repository can also be determined from the cache statistic `FreePages`. To obtain the free space, multiply `FreePages` by the page size, 16384.

9.2 Adding and Removing Extents

GemStone provides two ways to add extents:

- ▶ You can add new extents at startup by editing your GemStone configuration file and adding extent names and sizes to the `DBF_EXTENT_NAMES` (page 294) and `DBF_EXTENT_SIZES` (page 295) configuration options. Append the new values to the existing entries, just before the semicolon (;) delimiter. The new extents will be created the next time the Stone starts up.
- ▶ You can add extents while the Stone is running by invoking the Smalltalk methods described in this section. These methods are especially useful in avoiding or resolving low disk space conditions because the change takes effect immediately.

To Add an Extent While the Stone is Running

To prevent the repository from becoming full, you can dynamically add another extent to the Stone configuration. This section describes the Smalltalk methods that allow you to do this.

For general information about multiple extents, see “Configuring the Repository Extents” on page 44.

Possible Effects on Other Sessions

When a new extent is dynamically added to the logical repository through Smalltalk, sessions that are currently logged in must have access to the new extent. The possibility exists that an online session may terminate because it cannot open a new extent. Reasons for this condition could range from the inability to start a remote page server process to file permission problems.

CAUTION

The operating system creates the new extents with the ownership and permissions of the Stone repository monitor process. If these permissions are not the same as for other extents, you should use operating system commands to modify them as soon as possible. Such changes can be made without stopping the Stone.

A session’s view of which files make up the logical repository is updated whenever one of the following events occurs:

- ▶ Users commit or abort the session.
- ▶ The Stone repository monitor hands out disk resources to the session.

An explicit **commit** or **abort** may succeed but then cause the session to be terminated because of the inability to mount new extents immediately after the **commit** or **abort** operation.

Repository>>createExtent:

Privileges required: FileControl.

The Smalltalk method `createExtent:extentFilename` creates a new repository extent with the given extent file name (specified as a String). The new extent has no maximum size. The extent must be located on the machine running the Stone process; NFS-mounted disks are only allowed if your stone is configured to allow NFS mounted disks; see `STN_ALLOW_NFS_EXTENTS` (page 312). For example:

```
topaz 1> printit
SystemRepository createExtent: '$GEMSTONE/data/extent2.dbf'
%
```

You can execute this method when other users are logged in.

The Stone creates the new extent file, and it also appends the augmented extent list to your configuration file:

```
# DBF_EXTENT_NAMES written by Stone (user Bob) on 8/12/17 12:56:18
PDT
DBF_EXTENT_NAMES = "$GEMSTONE/data/extent0.dbf",
"$GEMSTONE/data/extent1.dbf",
"!TCP@mozart#dbf!/users/gemstone/data/extent2.dbf;"
```

If the given file already exists, the method returns an error and the specified extent is not added to the repository.

Creating an extent with this method bypasses any setting you may have specified for the `DBF_PRE_GROW` configuration option at system startup. Because extents created with this method have no maximum size and do not have an entry in a list setting for `DBF_PRE_GROW`, they cannot be pre-grown. If the repository is using weighted allocation, the new extent will be given a weight equal to the average weight of all other extents. (weighted allocation is discussed on page 47.)

If this method is run from a session on a host remote from the Stone, *extentFilename* must include a Network Resource String (NRS) specifying the Stone host. The syntax is shown above in the excerpt from the augmented configuration file. For information about NRS syntax, see Appendix C.

Repository>>createExtent:withMaxSize:

Privileges required: FileControl.

The Smalltalk method `createExtent:extentFilename withMaxSize:aSmallInteger` creates a new repository extent with the specified *extentFilename* and sets the maximum size of that extent to the specified size. You can execute this method when other users are logged in.

The size must be a non-zero positive integer representing the maximum physical size of the file in MB.

If the specified extent file already exists, this method returns an error and the extent is not added to the logical repository.

If `DBF_PRE_GROW` is set to `True`, this method will cause the newly created extent to be pre-grown to the given size. If the pre-grow operation fails, then this method will return an error and the new extent will not be added to the logical repository.

To Remove an Extent

The only way to remove an extent file is by first performing a Smalltalk full backup and restore to move the contents of that extent to other extents.

Privileges required: FileControl.

Reducing the number of existing extents requires special steps to ensure data integrity. If you must remove an extent file, follow this procedure:

Step 1. Back up your repository using the Smalltalk full backup procedure described under “How To Make a Smalltalk Full Backup” on page 215.

You cannot use an online or offline extent backup to remove or shrink extents (obviously).

Step 2. Shut down the Stone repository monitor.

Step 3. Modify the `DBF_EXTENT_NAMES` configuration option to show the new extent structure.

Step 4. Restore the repository from your Smalltalk full backup. Follow the GemStone restore procedure described under “Restoring from a Full Backup” on page 222.

9.3 Reallocating Existing Objects Among Extents

If you want to reallocate existing objects among two or more extents, the procedure depends in part on whether you are also changing the number of extents. Because changes to the `DBF_ALLOCATION_MODE` configuration option directly affect only the subsequent allocation of pages for new or modified objects, additional steps are necessary.

To Reallocate Objects Among a Different Number of Extents

If you are increasing or decreasing the number of extents and want to change allocation of existing objects as part of that operation, perform a Smalltalk full backup, then restore the backup after setting appropriate weights for `DBF_ALLOCATION_MODE`.

For example, suppose your existing repository contains 800 MB and you want to divide them about equally between the existing extent and a new one. To populate each extent with about 400 MB, follow this procedure:

Step 1. Back up your repository, using the Smalltalk full backup procedure described under “How To Make a Smalltalk Full Backup” on page 215. You cannot use an online or offline extent backup to redistribute objects (obviously).

Step 2. Shut down the Stone repository monitor.

Step 3. Modify the `DBF_EXTENT_NAMES` configuration parameter to show the new extent structure.

```
DBF_EXTENT_NAMES = $GEMSTONE/data/extent0.dbf,  
$GEMSTONE/data/extent1.dbf;
```

Step 4. Edit `DBF_ALLOCATION_MODE` to reflect the intended distribution of pages (see “Allocating Data to Multiple Extents” on page 47). For example:

```
DBF_ALLOCATION_MODE = 10, 10;
```

Step 5. Restore the repository from your Smalltalk full backup, using the procedure described under “Restoring from a Full Backup” on page 222.

If objects in the repository were explicitly clustered using instances of `ClusterBucket` that explicitly specified the first extent, those objects may tend to migrate back to that extent over time. You can prevent such migration by placing size limits on the existing extent, or by explicitly reclustered those objects in the new extent using a `ClusterBucket` that specifies either an `extentId` of `nil` or the `extentId` of the new extent. For information about clustering, refer to the *Programming Guide*.

To Reallocate Objects Among the Same Number of Extents

Changes to `DBF_ALLOCATION_MODE` directly affect only the subsequent allocation of pages for new or modified objects. When you restore into a repository with the same number of extents, the distribution of the original repository will be used in the restored repository, regardless of the `DBF_ALLOCATION_MODE`.

To change the allocation of existing objects, you can either restore into a repository with a different number of extents (as discussed in the previous section) or you can specify a temporary maximum size on the extent files during the restore, to force objects to be distributed as you want them.

For example, suppose your existing repository has three extents, and that you are running in sequential allocation mode. The first extent has 600 MB, while the second and third extents are 4 MB each (the minimum size). You now want to redistribute the objects so they are spread evenly over all three extents. You cannot simply change the `DBF_ALLOCATION_MODE`, and restore a backup into three extents; existing objects would be distributed according to the original allocation mode, that is, entirely in the first extent. Only new objects created after the restore would be created evenly over the three extents. To populate the three extents evenly, you can follow this procedure:

Step 1. Back up your repository, using the Smalltalk full backup procedure described under “How To Make a Smalltalk Full Backup” on page 215.

Step 2. Shut down the Stone repository monitor.

Step 3. Edit the `DBF_EXTENT_SIZES` configuration option to limit the size of the first extent temporarily to 200 MB. For example:

```
DBF_EXTENT_SIZES = 200MB, , ;
```

Step 4. Edit the `DBF_ALLOCATION_MODE` configuration option to reflect the intended distribution of pages (see “Allocating Data to Multiple Extents” on page 47). This setting determines the distribution of new or modified objects. For example:

```
DBF_ALLOCATION_MODE = 10, 10, 10;
```

Step 5. Restore the repository from your Smalltalk full backup, using the procedure described under “Restoring from a Full Backup” on page 222.

Step 6. If you want the first extent to grow beyond the temporary limit you set in Step 3, stop the Stone after you restore the repository. Edit the configuration file again, either specifying a higher limit or no limit. For example:

```
DBF_EXTENT_SIZES = , , ;
```

If objects in the repository were explicitly clustered using instances of `ClusterBucket` that explicitly specified the first extent, those objects may tend to migrate back to that extent over time. You can prevent such migration by maintaining the size limit set in Step 3, or by explicitly recluster those objects in the new extent using a `ClusterBucket` that specifies either an `extentId` of `nil` or the `extentId` of the new extent. For information, refer to the *Programming Guide*.

9.4 Shrinking the Repository

Privileges required: `SystemControl`, `GarbageCollection`, and `FileControl`.

To shrink the repository to its minimum size, make a Smalltalk full backup. Then take the repository offline and restore the backup into a copy of the GemStone distribution

repository. Use the following procedure, which compacts the repository into the minimum set of consecutive pages.

Step 1. Mark your repository for garbage collection, and wait for GemStone to complete the garbage collection and reclaim the space.

For example:

```
topaz 1> printit
SystemRepository markForCollection
%
```

The time required depends on several factors: the size of the repository, the number of Reclaim Gem sessions currently running, and (in multi-user mode) the status of other sessions.

If other users are logged in, space will not be reclaimed until all sessions have committed or aborted any transactions concurrent with the `markForCollection` process.

For further information on `markForCollection` and the garbage collection process, see Chapter 14, “Managing Growth”, starting on page 255. Details on the `markForCollection` method are under “MarkForCollection” on page 264.

Step 2. Make a Smalltalk full backup of your repository by sending it the message `fullBackupTo:` or `fullBackupTo: MBytes:`.

For example:

```
topaz 1> printit
SystemRepository fullBackupTo: '/users/backups/August_20'.
%
```

This example writes the backup to a single disk file. If you need to write multiple files, see “The `fullBackupTo: Methods`” on page 216.

Step 3. Take the repository offline:

```
topaz 1> printit
System shutDown
%
```

Step 4. Remove the existing repository extents. Obtain a copy of the distribution repository as the first (primary) extent by using the `copydbf` command. For example, assuming that all of your GemStone extents are in `$GEMSTONE/data`:

```
% cd $GEMSTONE/data
% rm extentNames
% copydbf $GEMSTONE/bin/extent0.dbf primaryExtentName
```

Use `chmod` to set the extent permission to what you ordinarily use for your repository.

Step 5. To put the repository back online, issue the `startstone` command:

```
% startstone -R gemStoneName
```

If you do not specify `gemStoneName`, `startstone` defaults to `gs64stone`.

Step 6. Log in to linked Topaz again.

NOTE

To perform the remaining parts of this procedure, you must be the only user logged in to GemStone. Logins will be disabled when you start the next step.

Step 7. Restore the repository by using the method

Repository>>restoreFromBackup:fileOrDevice, using the same file or device as in Step 2. Because it is being restored into a copy of the initial repository, the restored repository will be compressed to the minimum space. This example restores the backup from a single disk file:

```
topaz 1> printit
SystemRepository restoreFromBackup: '/users/backups/August_20'
%
```

If you need to restore multiple files, use the method

Repository>>restoreFromBackups:fileOrDeviceArray instead:

```
topaz 1> printit
SystemRepository restoreFromBackups:
#( '/users/backups/August_20.1'
    '/users/backups/August_20.2' )
%
```

(For more information, see “Restoring from a Full Backup” on page 222.)

GemStone reads the backup(s) and rebuilds the repository in a “shadow” object space that is invisible to users at this time. If the restore succeeds, GemStone commits the restore and returns a summary in the form of a nonfatal error message like the following:

```
Restore from full backup completed with 616227 objects
restored.
```

Each restore operation, on completion, terminates its GemStone session. You will need to log in again before performing the next restore operation.

Step 8. Between the time the full backup was started (Step 2) and the time the repository was shut down (Step 3), there may have been transactions on your repository. To ensure that no work is lost, restore from transaction logs and commit the restore. For example:

```
topaz 1> printit
SystemRepository restoreFromCurrentLogs
%
Restore from transaction log(s) succeeded.
```

Step 9. Commit the restore.

```
topaz> login
<details omitted>
successful login

topaz 1> printit
SystemRepository commitRestore
%
Restore from transaction log(s) succeeded. commitRestore
succeeded
```

9.5 Checking Page Fragmentation

Space within the repository is managed in pages having a fixed size of 16 KB. It is possible for these pages to become *fragmented*—that is, only partially filled with objects. GemStone automatically schedules reclaim of pages with greater than 10% free space as part of its garbage collection activity.

You can inquire about the amount of fragmentation in the repository by executing the following expression.

```
SystemRepository pagesWithPercentFree:aPercentage
```

Typical values of *aPercentage* range from 10 to 25. This method returns an array containing the following statistics:

- ▶ The total number of data pages processed
- ▶ The sum (in bytes) of free space in all pages
- ▶ The page size in bytes; always 16384.
- ▶ The number of data pages having at least the specified percentage of free space
- ▶ The number of data pages having at least the specified percentage of free space, that contain only a single object
- ▶ The total number of pages in the repository that contain only a single object
- ▶ The number of pages that should be in the `scavengablePages` but are not.

`pagesWithPercentFree`: executes using the multi-threaded scan. See “Multi-Threaded Scan” on page 278 for details.

9.6 Disk Space and Commit Record Backlogs

Sessions only update their view of the repository when they commit or abort. The repository must keep a copy of each session’s view so long as the session is using it, even if other sessions frequently commit changes and create new views (commit records). Storing the original view and all the intermediate views uses up space in the repository, and can result in the repository running out of space. To avoid this problem, all sessions in a busy system should commit or abort regularly.

For a session that is not in a transaction, if the number of commit records exceeds the value of `STN_CR_BACKLOG_THRESHOLD`, the Stone repository monitor signals the session to abort by signaling `TransactionBacklog` (also called “sigAbort”). If the session does not abort, the Stone repository monitor terminates it.

Sessions that are in transaction are not subject to losing their view forcibly. Sessions in transaction enable receipt of the signal `TransactionBacklog`, and handle it appropriately, but it is optional. It is important that sessions do not stay in transaction for long periods in busy systems; this can result in the Stone running out of space and shutting down. However, sessions that run in automatic transaction mode are *always* in transaction; as soon as they commit or abort, they begin a new transaction. (For a discussion of automatic and manual transaction modes, see the “Transactions and Concurrency Control” chapter of the *Programming Guide*.)

To avoid running out of disk space, we recommend that you use *manual transaction mode* whenever possible. To enter manual transaction mode:

```
topaz> printit
System transactionMode: #manualBegin
%
```

At the point that this session needs to commit a change, begin a transaction manually, then make the changes:

```
topaz> printit
System beginTransaction .
AllUsers addNewUserWithId: #Jane password: 'gemstone' .
System commitTransaction
%
```

After you commit (or abort) the transaction, your session will return to waiting outside of a transaction.

Handling signals indicating a commit record backlog

Even in manual transaction mode, it is possible to cause a commit record backlog, depending on how your system is configured. Sessions should ensure that they commit or abort regularly, or set up sigAbort handlers to abort when requested by the Stone. A sigAbort handler may be as simple as this:

Example 9.2 sigAbort handler

```
Exception
  installStaticException:
  [ :exception :GSdictionary :errID :array |
    System abortTransaction.
    System enableSignaledAbortError).
```

Note that a session that is entirely idle does not become aware of the signal to abort, and may timeout and be terminated by the stone in spite of the handler. If your application may have idle sessions, we recommend setting up a timer that causes regular aborts when the session is otherwise idle.

Sessions that are in transaction, and therefore immune from the sigAbort mechanism, may also be signaled when there is a commit record backlog. When the number of commit records exceeds the value of `STN_CR_BACKLOG_THRESHOLD`, and the session holding the oldest commit record is in transaction, the Stone repository monitor signals the session by sending `TransactionBacklog`. The session then has the opportunity to perform a `continueTransaction` to update its view of unmodified objects. It may also commit or abort. Unlike sigAbort, the session can choose to ignore this message and will not receive further signals from the stone.

Example 9.3 finishTransaction handler

```

Exception
  installStaticException:
  [ :exception :GSdictionary :errID :array |
    System continueTransaction.
    System enableSignaledFinishTransactionError).

```

For more information on these signals, see the *Programming Guide for GemStone/S 64 Bit*.

9.7 Recovering from Disk-Full Conditions

The Stone repository monitor has two critical needs for disk space. It must be able to:

- ▶ Append to the transaction log as sessions commit changes.
- ▶ Expand the repository as necessary to allocate free pages to current sessions or to sessions logging in.

Whenever the Stone cannot log transactions or cannot find sufficient free space in the repository, it issues an error message to any session logged in as DataCurator or SystemUser. If users report that GemStone appears to be hung or that they get a disk-full error while logging in, you should check one of these administrative logins for such a message. The message is also written to the Stone's log file.

The following topics explain the Stone's actions in greater detail and describe steps you can take to provide sufficient space.

For details on how tranlog full conditions are handled, see "Recovering from Disk-Full Conditions" on page 196.

Repository Full

The Stone takes a number of actions to prevent the repository from becoming completely full. If the free space remaining in the repository falls below the level set by the STN_FREE_SPACE_THRESHOLD configuration option and the Stone cannot allocate more space in any extent, it takes the following actions to prevent a system crash:

1. It becomes more aggressive about disposing of commit records so that garbage collection can proceed. (If the Stone is very busy, a backlog of commit records can accumulate.)
2. It starts a checkpoint if there isn't one in progress and reduces the checkpoint interval to three minutes until the condition clears. (This checkpoint may free pages that have been reclaimed.)
3. It writes a message to the Stone log to indicate the condition.
4. It prevents new logins except for DataCurator and SystemUser accounts. It issues a disk-full error to other sessions attempting to log in.
5. It signals the Exception RepositoryError to any sessions logged in (or logging in) as DataCurator or SystemUser so that they know disk space is becoming critical.

6. It signals Gem session processes to return all except five free pages per extent. It responds to requests for additional pages by allocating only five pages at a time.
7. If the free space available drops below 400 KB (50 pages), the Stone stops responding to page requests from sessions that are not logged in as DataCurator or SystemUser. This action prevents users from acquiring all of the available space, which would cause the system to crash. Gem session processes appear to “hang” while they are waiting for pages. The unhonored page requests are granted when the free space goes back above the threshold.
8. If the previous steps do not solve the problem within the time specified by the `STN_DISKFULL_TERMINATION_INTERVAL` configuration option, then the Stone begins to terminate sessions holding on to the oldest commit record *even if the session is in a transaction*. This action applies to any user session, including logins as SystemUser and DataCurator. Allowing the Stone to dispose of the commit record allows additional garbage collection.

NOTE

You can configure the Stone to never terminate sessions by setting `STN_DISKFULL_TERMINATION_INTERVAL` to 0; however, doing so increases the risk of GemStone shutting down because of a lack of free space in the repository.

9. When the condition clears, another message is written to the Stone log and operation returns to normal.

If you see a message like the following while logged in as DataCurator or SystemUser or in the Stone log, disk space is becoming critical:

```
The repository is currently running below the freeSpaceThreshold.
```

When the system must dynamically expand the repository, it checks one extent at a time, in the order dictated by the allocation strategy, to see if that extent can be expanded to create more space. When no extents can be extended and the free space is below `STN_FREE_SPACE_THRESHOLD`, the Stone takes the actions described above.

Failure to expand an extent has two possible causes: either the disk containing the extent is full, or the extent has reached its maximum size as set by the `DBF_EXTENT_SIZES` configuration option.

There are a number of things you can do to create more space in an existing extent, or you can create a new extent. Each of these actions may create sufficient additional space for immediate needs:

- ▶ Warn the current users about the problem, and have them log out until enough space is made available.
- ▶ Remove any non-essential files to create enough space for expanding the repository.
- ▶ Create a new extent through Smalltalk with `Repository>>createExtent:` or a related method. If the Stone has stopped, you can create a new extent by editing the parameters in the configuration file before restarting it. See “Adding and Removing Extents” on page 187.

Managing Transaction Logs

Transaction logs hold records of changes to the repository, which allow you to recover any changes made between backups. This chapter describes how to manage these transaction logs.

Overview (page 199)

explains transaction logs, how to configure them, and how to use them when restoring from backup.

How To Manage Full Logging (page 203)

describes transaction log management when transaction logs are configured for full logging.

How To Manage Partial Logging (page 207)

describes transaction log management when transaction logs are configured for partial logging.

How To Recover from Tranlog-Full Conditions (page 208)

describes how the system responds when there is no disk space for transaction logs, and what to do in these circumstances.

10.1 Overview

A transaction log contains the information necessary to redo transactions to the repository that have been committed by GemStone sessions since the last checkpoint or orderly shutdown. This log is used to recover from crashes such as those caused by a power failure, an operating system failure, or the killing of GemStone monitor processes.

If you need to restore the repository from a backup, transaction logs written in full-logging mode can be used to recreate all transactions committed since the most recent backup was written.

The transaction log is implemented as a sequence of files having names of the form `tranlog1.dbf ... tranlogNNN.dbf`. The numeric `fileId` starts at 1 when the Stone starts with a copy of the initial repository extent (`$GEMSTONE/bin/extent0.dbf`), and there are no existing transaction logs with that `tranlog` ID in any transaction log directories.

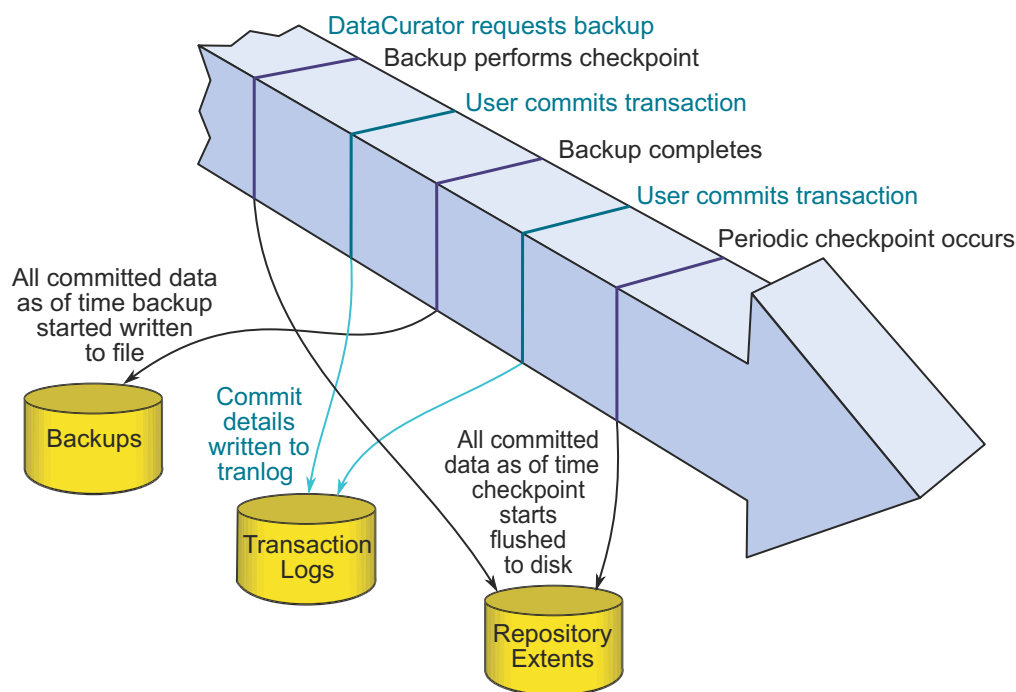
If the Stone starts on an existing repository without any logs present, the `fileId` will be one greater than when the repository was last shut down cleanly.

The filename prefix, by default “`tranlog`”, can be controlled by setting the `STN_TRAN_LOG_PREFIX` configuration option.

The transaction logs are written to a list of directories or raw partitions specified by the `STN_TRAN_LOG_DIRECTORIES` configuration option, which is treated as a circular list. Each log is limited to the size set for that directory or raw partition by the `STN_TRAN_LOG_SIZES` configuration parameter. When one log is full, logging switches to the next directory or raw partition. (What happens when logs have been created in all directories is discussed in Table 10.1 on page 201) Collectively, the transaction log files logically form an extremely large sequential file with a maximum size of 4×10^6 GB.

As users commit changes to the repository, GemStone writes the changes to a transaction log. Periodically, the stone repository monitor performs a checkpoint, at which point all committed changes are guaranteed to be written to the extents. During the periods between checkpoints, the transaction logs hold the record of the changes, so they are available if an unexpected shutdown occurs.

Figure 10.1 Normal Operation



Use ordinary UNIX utilities to backup the transaction logs in the file system. To backup a transaction log that is on a raw disk partition, use **copydbf** to copy it to a file system or to a tape device. You’ll also need to use **removedbf** to clear the partition for reuse.

Logging Modes

GemStone provides two modes of transaction logging, selected by setting the `STN_TRAN_FULL_LOGGING` configuration option:

- ▶ To provide real-time incremental backup of the repository, set `STN_TRAN_FULL_LOGGING` to True. All transactions are logged regardless of their size. This mode is strongly recommended for deployed GemStone systems.
- ▶ To allow a simple operation without critical data to run unattended for an extended period, set `STN_TRAN_FULL_LOGGING` to False. This mode, known as *partial* logging, provides automatic recovery from system crashes that do not corrupt the repository, such as fatal errors or loss of power. However, if you do experience extent corruption and need to restore from backup, you cannot recover any changes made after the backup was taken.

Table 10.1 compares full and partial transaction logging.

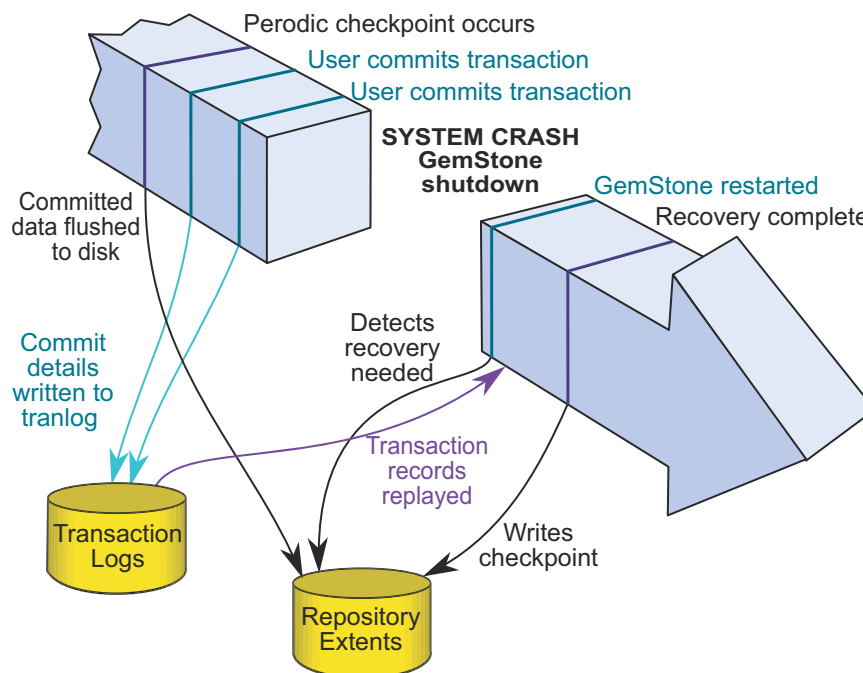
Table 10.1 Comparison of Full and Partial Transaction Logging

Characteristic	Full Logging <code>STN_TRAN_FULL_LOGGING = TRUE</code>	Partial Logging <code>STN_TRAN_FULL_LOGGING = FALSE</code>
Type of transaction logged	All transactions	Only those transactions smaller than <code>STN_TRAN_LOG_LIMIT</code> ; successful commits of larger transactions cause an immediate checkpoint
Recovery from system crash (extents are OK)	Yes, automatic recovery during restart using checkpoint and log	Yes, automatic recovery during restart using checkpoint and log
Replay of transactions since last backup (as after media failure)	Yes – can carry forward GemStone backup by recreating subsequently committed transactions	No – cannot replay transactions since the backup
Action when current log is full	Logging moves to the next directory or to the head of the list. If a file system directory, the Stone opens a new log file, and retains existing transaction logs. If a raw partition, a new log can be opened only if the previous one has been removed. There is no limit on the number of file system logs in the directory, but the number of raw partition logs is limited to the number of partitions listed in <code>STN_TRAN_LOG_DIRECTORIES</code> .	Logging moves to the next directory or to the head of the list. The Stone removes the existing transaction log before opening a new one. The maximum number of logs on line at one time depends on the number of directories or raw partitions in the list.
Action when log space becomes full	The Stone pauses execution if it cannot find space in any of the specified directories or raw partitions.	The Stone deletes log files from the circular list of directories and keeps running.
Administrative task	Monitor log space; archive log files and delete them as necessary	None

In the event of a system crash, GemStone can recover by automatically replaying transactions recorded in the transaction log, from the latest checkpoint before the crash to the end of the log at the moment of the crash (Figure 10.2). This allows you to restart from where you were at the time of the crash.

No special administrative action is required to perform this recovery. Every time the system does a normal startup, it checks to see if there are transactions that need to be recovered. Thus, if the extents and transaction logs are good, the system will automatically recover from a crash.

Figure 10.2 System Crash



Restoring Transactions to a Backup

An important use of transaction logs is to restore transactions that were committed between the last full backup and a system failure. If you experience system failure and your current extents are no longer usable, you can still recover all data, provided that the repository already is in full transaction logging mode and that the backup was made in that mode.

For details on restoring transactions to a backup, see "How to Restore Transaction Logs" on page 230.

How the Logs Are Used

You can only use transaction logs to restore your system from backup if you are in full logging mode.

After creating a backup, you need to retain all the transaction logs that are created during and after the backup. To determine the oldest transaction log that will be required, use the `copydbf` utility to query the backup file. For example:

```
unix> copydbf -i backup.dat
Source file: backup.dat
  file type: backup  fileId: 0
  byteOrder: Intel (LSB first)  compatibilityLevel: 850
  Full backup started from checkpoint at: 08/20/17 17:16:35
PDT.
  Oldest tranlog needed for restore is fileId 4 ( tranlog4.dbf
).
Backup was created by GemStone Version: 3.4.0 .
```

For this example, `tranlog4.dbf` and later must be available. If any of these transaction logs are deleted or lost, you will not be able to recover completely. These transaction logs may be archived elsewhere, as long as they can be readily be made available if you do need to restore from backup.

If you do need to restore your system, first restore from backup using the using the appropriate procedure, then replay transactions committed since the backup by reading the transaction logs in the order in which they were created.

For detailed procedures on restoring from backup and replaying transaction logs, see the instructions in Chapter 11, “Making and Restoring Backups”, starting on page 209

NOTE

Restoring a repository from backup resets it - both the GemStone kernel and your application classes and data - to the state it was in at the time the backup was created. Anything that was done after that can be recovered only by replaying transaction logs in order, or by restoring a more recent backup.

10.2 How To Manage Full Logging

When the system is operating with the `STN_TRAN_FULL_LOGGING` configuration option set to `True`, you (as system administrator) should monitor the available log space. If the log space defined by `STN_TRAN_LOG_DIRECTORIES` becomes full, users will be unable to commit transactions to the repository until space is made available.

For transaction logs in file system directories, “full” means that there is no free space in the file systems containing those directories.

For transaction logs in raw partitions, “full” means that all partitions listed already contain a GemStone transaction log or other repository file. After archiving an existing log, you must invoke `removedbf` (page 346) before that partition can be reused.

There are two recovery situations to consider in managing transaction logs under full logging:

- ▶ Recovery from a system crash requires logs for all transactions committed since the last *checkpoint*. Because of the way GemStone logs changes involving large objects, parts of these transactions may be in earlier logs. The method `Repository>>oldestLogFileIdForRecovery` returns the `fileId` of the oldest log that would be needed if a crash were to occur at that point. All logs needed for crash recovery should be kept online.

- ▶ Recovery from damaged extents, such as a media failure, requires all transaction logs since the last *backup*, and earlier logs may be needed if lengthy transactions were in progress at the time the backup started. Log files not needed for crash recovery may be archived offline, although restoring them will take longer.

To Archive Logs

Ordinary UNIX tools, such as **tar** and **cp**, can be used to move log files to other locations or to archival media. We recommend that you archive and free one complete log directory at a time, in the order listed in the `STN_TRAN_LOG_DIRECTORIES` configuration option.

NOTE

*If you must rename the log files, we recommend that you preserve the digits in the original filename as an aid to restoring the files in sequence should that become necessary. If your transaction logs are in raw disk partitions, **copydbf** adds the fileId when you copy a log to a file system directory.*

Two special commands are provided for working with raw disk partitions:

- ▶ The **copydbf** command copies a repository file (extent, transaction log, or full backup) to or from a raw disk partition. If the destination is a directory in the file system, **copydbf** generates a filename that includes the file type and its internal fileId.
- ▶ The **removedbf** command writes over a raw partition so that GemStone will no longer think it contains a repository file.

To determine the current size of a transaction log that is in a raw partition, use the method `Repository>>currentTranlogSizeMB`. This method returns the log size (in MB) as an Integer.

To determine the oldest transaction log that would be needed to recover from the most recent checkpoint, use the method `Repository>>oldestLogFileIdForRecovery`. This method returns the internal fileId, which is part of the filename for transaction logs in the file system. If a session was in a lengthy transaction at the time of a system crash, the oldest log needed during recovery may be one that was written before the last checkpoint occurred.

You can obtain similar information by applying **copydbf -i** to an extent. For example:

```
% copydbf -i extent0.dbf
Source file: extent0.dbf
  file type: extent  fileId: 0
  byteOrder: Sparc (MSB first) compatibilityLevel: 844
  Last checkpoint written at: 09/06/17 15:30:30 PDT.
  Oldest tranlog needed for recovery is fileId 5 ( tranlog5.dbf ).
  Extent was shutdown cleanly; no recovery needed.
```

To determine the oldest transaction log needed to roll forward from a backup, apply **copydbf -i** to the backup:

```
% copydbf -i backup.dat
Source file: backup.dat
  file type: backup  fileId: 0
  byteOrder: Sparc (MSB first) compatibilityLevel: 850
  Full backup started from checkpoint at: 09/06/17 15:28:37 PDT.
  Oldest tranlog needed for restore is fileId 5 ( tranlog5.dbf ).
  Backup was created by GemStone Version: 3.4.0 .
```

For an example script showing how to archive transaction logs out of raw partitions, see `$GEMSTONE/examples/archivelog.sh`. You will need to edit the script to conform to your own partition names and archive location, and then test it.

To Add a Log at Run Time

Privileges required: FileControl.

You can add a directory or a raw partition for transaction logs to the existing list without shutting down the Stone repository monitor. When you do this, the repository monitor also records the change in its configuration file so that the addition becomes permanent. Send the following message:

```
SystemRepository addTransactionLog: deviceOrDirectory size: aSize
```

For example:

```
topaz 1> printit
SystemRepository addTransactionLog: '/users/tlogs2' size: 100
%
```

The argument *aSize* sets the maximum log size (in MB) for *deviceOrDirectory*. It will be added to the list in `STN_TRAN_LOG_SIZES` (page 331).

You can use the method `System class>>stoneConfigurationAt:` to examine the contents of `STN_TRAN_LOG_DIRECTORIES` at run time. For information, see “How To Access the Server Configuration at Run Time” on page 55. The Repository methods in Table 10.2 return other information that is helpful in managing transaction logs.

Table 10.2 Repository Methods for Information About Transaction Logs

Method	Description
<code>currentLogDirectoryId</code>	Returns a positive <code>SmallInteger</code> , which is the one-based offset of the current log file into the list of log directory names.
<code>currentLogFile</code>	Returns a <code>String</code> containing the name of the transaction log file to which records currently are being appended.
<code>currentTranlogSizeMB</code>	Returns an <code>Integer</code> that is the size (in MB) of the currently active transaction log.
<code>allTranlogDirectories</code>	Returns an <code>Array of Strings</code> corresponding to the <code>STN_TRAN_LOG_DIRECTORIES</code> configuration.
<code>allTranlogSizes</code>	Returns an <code>Array of SmallIntegers</code> corresponding to the <code>STN_TRAN_LOG_SIZES</code> configuration.
<code>logOriginTime</code>	Returns the log origin time of the receiver, the time when a new sequence of log files was started. For details, see the method comment in the image.

Table 10.2 Repository Methods for Information About Transaction Logs

Method	Description
<code>oldestLogFileIdForRecovery</code>	Returns a positive <code>SmallInteger</code> , which is the internal <code>fileId</code> of the oldest transaction log needed to recover from the most recent checkpoint, if the Stone were to crash as of now.
<code>restoreStatusOldestFileId</code>	Returns a <code>SmallInteger</code> , which is the internal <code>fileId</code> of the oldest transaction log needed for the next restore from log operation.

To Force a New Transaction Log

Privileges required: `FileControl`.

You can force closure of the current log and opening of a new log at almost any time by using the method `Repository>>startNewLog`. The method performs the following sequential actions:

1. Starts a checkpoint.
2. Waits until the checkpoint completes.
3. Starts the new log.
4. Returns a `SmallInteger`, which is the `fileId` of the new log.

In the following example, the new transaction log file would be `tranlog9.dbf`.

```
topaz 1> printit
SystemRepository startNewLog
%
_
```

If a checkpoint is already in progress when you execute `startNewLog`, the method will fail and return `-1` instead. If you're using this method in an application, therefore, you need to accommodate the possibility of such a failure with code such as:

```
| id |
id := SystemRepository startNewLog.
[ id < 0 ] whileTrue: [
  System sleep: 1.
  id := SystemRepository startNewLog ].
```

To Initiate a Checkpoint

Privileges required: `SystemControl`.

`System` class provides two methods that you can use to start checkpoints manually. These methods do not commit, abort, or otherwise modify the current transaction.

`startCheckpointSync`

Starts a new checkpoint and returns when the new checkpoint has completed. If a checkpoint is already in progress, this method waits until the current checkpoint completes, then starts a new checkpoint. Returns `true` if a new checkpoint was successfully completed, returns `false` if a new checkpoint could not be started because transaction logs are full or checkpoints are suspended.

`startCheckpointAsync`

Starts a new checkpoint if a checkpoint is not already in progress and returns immediately, without waiting for the checkpoint to finish. Returns true if a checkpoint is in progress or was started; returns false if a checkpoint could not be started because transaction logs are full or checkpoints are suspended.

To Change to Partial Logging

Once the full transaction logging has been started on a repository, the `STN_TRAN_FULL_LOGGING` state of True persists regardless of later changes to the configuration file.

To terminate full logging, the procedure is:

- Step 1.** Make a Smalltalk full backup (not an extent snapshot), following the instructions under “How To Make a Smalltalk Full Backup” on page 215.
- Step 2.** Edit the configuration file to set the `STN_TRAN_FULL_LOGGING` option to False.
- Step 3.** Stop the Stone repository monitor, and restore the backup following the instructions under “Restoring from a Full Backup” on page 222. Be sure to restore into a copy of the empty distribution extent (`$GEMSTONE/bin/extent0.dbf`).
- Step 4.** Restart GemStone.

10.3 How To Manage Partial Logging

Partial logging provides ease of administration along with some protection against loss of data from system crashes. The Stone repository monitor treats the log directories as a circular list. If the file in the current directory reaches the limit set by `STN_TRAN_LOG_SIZES`, the Stone switches to the next directory in the list that does not contain a transaction log. In the process of creating log n , the Stone attempts to find and delete log $(n - \text{size_of_STN_TRAN_LOG_DIRECTORIES})$; for example, if the new log will be `tranlog7.dbf` and there are three elements in `STN_TRAN_LOG_DIRECTORIES`, the Stone searches all three in attempting to delete `tranlog4.dbf`.

If there is only one directory specified in by the `STN_TRAN_LOG_DIRECTORIES`, then the stone deletes the log file in this directory before starting a new log. This means that badly-timed crashes may not be recoverable. You should ensure that there are two directories specified and that there is sufficient disk space for at least two log files, so that one can be preserved when the next is opened.

To Change to Full Logging

To change a repository from partial to full logging, simply change the `STN_TRAN_FULL_LOGGING` (page 329) setting to True and restart the Stone repository monitor.

Be sure to make a new GemStone full backup in full-logging mode so that you will be able to restore from the transaction logs if necessary. Transaction logs cannot be restored to backups that were made in partial-logging mode.

10.4 How To Recover from Tranlog-Full Conditions

Transaction Log Space Full

If the space for transaction logs becomes full, the Stone stops processing commits or other requests that initiate a write to the transaction log. Sessions performing these operations are blocked until the condition is resolved and may appear to the user to be hung. The Stone writes a message like the following in its log:

```
The tranlog directories are full and the stone process is waiting
for an operator to make more space available by either cleaning up
the existing files (copying them to archive media and deleting
them) or by adding a new tranlog directory.
```

Also, the `Exception RepositoryError` is signaled in any sessions that have enabled receipt of this error by sending `System enableSignalTranlogsFull`, and setting up a handler for this error.

Once enabled, you can disable receipt of this error by sending `System disableSignalTranlogsFull`, and determine the current status by `System signalTranlogsFullStatus`.

If the transaction log space is full, you have the following options:

- ▶ You can free space by taking some existing log files offline. Archive them using operating system utilities and then remove them. GemStone can reuse that slot in the circular list of log directories. (To archive and remove a log file from a raw partition, use `copydbf` and then `removedbf`.)
- ▶ You can increase the available log space by adding a raw partition or a directory on another disk drive to this list specified for the configuration option `STN_TRAN_LOG_DIRECTORIES`. Add its maximum file size to `STN_TRAN_LOG_SIZES`. For information on how to make these changes while GemStone is running, see “To Add a Log at Run Time” on page 205.

When transaction log space becomes available, waiting sessions can complete operations that were blocked.

Making and Restoring Backups

This chapter describes how to make backups of your GemStone/S 64 Bit repository and how to use the backups and transaction logs to restore the repository.

Overview (page 209)

explains the importance of backups.

Types of Backups (page 211)

describes the different types of backups, and how to choose a backup strategy.

How To Make an Extent Snapshot Backup (page 212) and **How To Make a Smalltalk Full Backup** (page 215)

describe the processes for making backups.

How to Restore from Backup (page 219) and **How to Restore Transaction Logs** (page 230)

describe how to restore a backup and any subsequent transaction logs, to reproduce a complete repository state.

How to Make and Restore Secure Backup (page 225)

describes the process of making a digitally signed and optionally encrypted backup.

Special Cases and Errors in Restore (page 232)

provides additional information for special cases of restore.

11.1 Overview

To safeguard your repository, you should create a backup of your GemStone repository periodically, and store the backup in a safe place. Backups provide security in case of problem with power, operating system, disks, or other system corruption, and if used in combination with transaction logs, preserve all committed data against loss.

Making a backup of the GemStone repository captures the state of the system at a particular moment in time, and restoring that backup can return your system to the state it was in at the time the backup started. A GemStone backup is a backup of not only your application data, but also your application code and GemStone kernel code, and of user profiles and passwords and so on - everything in the repository. Because the backup

includes kernel code, backups can only be restored into the same version of GemStone as that in which the backup was created; otherwise the kernel classes and methods may not be appropriate for that version.

Between these periodic backups, transaction logs capture all committed changes that occur in the repository (provided the repository is in full logging mode). By preserving the backup and a set of transaction logs, you have the ability to recreate the system up to the last committed change in the transaction logs.

In partial tranlog mode, the transaction logs cannot be applied after restoring a backup. In this case, the transaction logs are useful when recovering from transient problems such as unexpected shutdown, but restoring a backup can only restore the system to the state it was in at the time of the backup. Later transaction logs in partial logging mode cannot be applied to recover work done after the time of the backup.

You should establish a regular backup process and schedule that fits your application requirements, and a system of managing and archiving the backup files and transaction logs that will allow you to recover smoothly after any problems.

In addition to regular backups, to ensure protection from disk failure, we recommend that you either use mirrored disks or operating system mirroring. For more information, see “Developing a Failover Strategy” on page 37.

Warm and Hot Standbys

GemStone’s backup and restore mechanisms can be used to set up a secondary server, running in parallel with the primary server and ready to take over as quickly as possible in case of any failure of the primary system.

To do this, a backup of the primary server is restored into a separate location. This backup stays running in restore mode, and as transactions are generated on the primary server, they are restored into the standby system. In case of failure of the primary system, the standby can be quickly ready to use and in a state identical to the failed system.

For details on how to set up a warm or hot standby system, see Chapter 12, “Warm and Hot Standbys”, starting on page 239.

Version Compatibility

It is not always possible to restore backups made by a previous version of GemStone into a new version. Since kernel classes and methods are also included in the full backup, restoring an older version will result in GemStone Smalltalk code that is not correct for the GemStone version.

If you archive backups of your GemStone repository over multiple upgrades of your GemStone installation, you should also archive the GemStone executables for each version.

While not supported, in cases where it is possible to restore the backup from an older version into a more recent version, you should follow the upgrade instructions in the *Installation Guide* to run `upgradeImage`, to ensure kernel code is updated.

11.2 Types of Backups

GemStone supports several types of backup:

- ▶ Offline extent snapshot backups
- ▶ Online extent snapshot backups
- ▶ Smalltalk full backups
- ▶ Smalltalk secure backups

Extent snapshot backups consist of operating system copies of the extent files.

When the repository is offline, and was cleanly shut down, the extent files can be copied using regular OS copy functions with no further considerations.

To make extent file copies of a repository that is in use (online), checkpoints must be suspended for the duration of the extent copy. The extents are updated during checkpoints, so if a checkpoint occurs during extent file copy, it is likely the backup files will be corrupted and unusable.

Smalltalk full and secure backups are made by executing backup methods in GemStone code. These can only be created when the system is running. Executing the backup methods will cause all live objects in the repository as of the time the backup method execution began to be written out to one or more operating system files. Dead objects, and internal structures such as the object table, are not written out, so these files typically are somewhat smaller than the repository extent size (excluding free space in extents).

Determining which type of backup to make depends on the size of your repository and the uptime requirements.

- ▶ **Offline extent backups** are the most simple, since nothing is needed beyond clean shutdown and file copy. However, since these must be taken when the repository is shut down, they are not suitable for systems that must be available 24x7.
- ▶ **Online extent snapshots** require the most effort to setup, since checkpoints must be suspended for the entire duration of the file copy. Since file copy is limited only by throughput of the physical disks, for large repositories that are in use 24x7, online extent backups will have the smallest impact on availability and on performance. They will also be much faster to restore than full backups.
- ▶ **Full backups** are convenient to run in highly-available systems that are not shut down regularly. However, backup execution places load on the system, and should be avoided during periods of heavy system use. Restore from full backup will also take much longer than offline or online extent backups.

Backups made using fullBackup methods have other uses; restoring these backup files can be used to change the number of extents, redistribute objects among extents, or reduce the size of extent files. Full backups can be written directly in compressed form, for the most efficient storage of the resulting backups.

- ▶ **Secure backups** include both digitally signed backups, and backups that are encrypted as well as signed. Different methods are used to create and restore these backups, but otherwise they are similar to full backups.

Secure backups are useful when you want to verify that a backup has not been modified, to archive backups to a non-secure location, and to distribute backups privately to multiple destinations without having to share private keys.

Full vs. Partial Transaction logging

As described under “Logging Modes” on page 201, your repository may be run in partial transaction logging mode, or in full transaction logging mode.

In partial transaction logging mode, you cannot make online extent backups, since checkpoints cannot be suspended while you are in this mode.

While you can make Smalltalk full backups, or offline extent copy backups, you cannot restore transaction logs into these backups. If you need to restore from backup, any work done after the start of the backup is permanently lost. For repositories with valuable data, we recommend that you run in full logging mode to avoid data loss in case of extent corruption.

Verify Backup Process

Creating a backup and archiving transaction logs is only useful if you can restore them successfully in case of a system failure. To make sure that your procedures for archiving and restoring backups is complete and correct, it is good practice to perform the restore operation into a non-production system, replay tranlogs, and audit the restored repository. Instructions for auditing can be found under “Repository Page and Object Audit” on page 132.

Performing this exercise ensures that if you do have an emergency situation, you will have the required files available and be familiar with the process of restore, and avoid the risk of losing data.

11.3 How To Make an Extent Snapshot Backup

Extent snapshot backups are file system copies of the repository extents. These copies can be made when the repository is not running (offline); or when the repository is running (online), provided you suspend checkpoints for the duration of the extent file copy.

WARNING

File system copies of the extents of a running GemStone repository that are taken during a period that includes a checkpoint will have inconsistent state, and not be usable for restore.

Extent Snapshot Backup when the Repository is shutdown

When the repository is shut down, you can safely perform a file system backup of the extents files. During the shutdown process, a checkpoint is performed in which all committed transactions are written to the extents. A copy of the extents after an orderly shutdown constitutes a complete operating system backup of the repository without requiring any transaction logs.

If GemStone was not shut down cleanly, file system copies of the extents are usable, but they will not include any transactions committed since the last completed checkpoint before the shutdown. In order to recover later work, you will also need one or more transaction logs.

This applies for both partial logging and full logging.

copydbf -i will report if the extents were cleanly shutdown and the oldest tranlog required for recovery if the extents were not cleanly shutdown.

Extent Snapshot Backup when the Repository is running

When the repository is running, you must suspend checkpoints before starting the extent file copy, and resume checkpoints when the file copy is complete.

You should not attempt to take online extent snapshot backups when the repository is in partial logging mode (`STN_TRAN_FULL_LOGGING = FALSE`), since checkpoints cannot be suspended in partial logging mode.

Three steps are involved in an online extent backup

1. Suspend checkpoints.

Checkpoints are not permitted while the extent file are being copied for the online backup. There must not be a checkpoint in progress when the first extent file copy starts, and no checkpoints are allowed to begin until the last extent file copy has completed. All other database operations (including commits, aborts, and the creation of new tranlogs) are permitted during the online extent snapshot backup.

To suspend checkpoints for a specified number of minutes, call `System class >> suspendCheckpointsForMinutes:.` If this method is called while a checkpoint is already in progress, it will block until the current checkpoint completes. On some systems under heavy load, checkpoints may take some time to complete; the period in which checkpoints are suspended does not begin until the previous checkpoint is complete.

If one session attempts to suspend checkpoints and is blocked while the current checkpoint completes, and then a second session attempts to suspend checkpoints, the second session fails and the method returns false.

If the system is shut down while checkpoints are suspended, checkpoints will be re-enabled and a final checkpoint will be written during the clean shutdown process. Any extent snapshot backups in progress during system shutdown must be discarded.

To query the current status of checkpoints, call `System class >> checkpointStatus.` This method returns an Array object containing a Boolean that indicates whether checkpoints are suspended, and an Integer giving the number of seconds remaining in the suspension.

Example 11.1 Suspending Checkpoints

```
topaz 1> printit
System checkpointStatus
%
an Array
  #1 false
  #2 0

topaz 1> printit
System suspendCheckpointsForMinutes: 15
%
true

topaz 1> printit
System checkpointStatus
%
an Array
  #1 true
  #2 900
```

We recommend using a value of *minutes* that is much larger than any possible anticipated time, taking into consideration the amount of time backups may take after future repository growth. If checkpoints resume before the extent/s copy is complete, the snapshot will not be usable.

It is preferable to have checkpoints suspended for as short a time as possible, but it is safer for the backup script to manually resume checkpoints after the file copies are completed, rather than relying on tuning the time out period.

2. Copy the repository extents.

Once checkpoints are suspended, the session requesting the suspension can log out from GemStone and start the extent copy, using operating system commands or `copydbf`.

3. Resume checkpoints.

Once the extent copy has completed, a session should log in to GemStone and request the Stone to resume checkpoints (`System class >> resumeCheckpoints`). The result of this method is `false` if checkpoints were not previously suspended before executing `System class >> suspendCheckpointsForMinutes:` (as in Step 1), and `true` if they were previously suspended.

```
topaz 1> printit
System resumeCheckpoints
%
true
```

From this result, you can determine if the online extent backup was completed while checkpoints were still suspended. If the backup was completed in time, no further action is required and the backup is complete. If the backup did not complete before

checkpoints were resumed, then the backup must be discarded and another online extent backup must be taken.

CAUTION

Make sure your backup code checks this result, since a false return value means that your backup is not usable.

An Example Script

The GemStone installation directory includes an example script `$GEMSTONE/examples/admin/onlinebackup.sh`. You can customize this script for your own system.

This script does not include code to make file system copies of the extents; you must add the necessary code to perform this task. This script provides a default checkpoint suspension of 15 minutes, which may or may not be sufficient time.

NOTE

The example script `onlinebackup.sh` is unsupported. It is provided here for your convenience, and is subject to change in future releases.

Be sure to review and test your script adequately to ensure the integrity of your backups.

11.4 How To Make a Smalltalk Full Backup

You can create a backup of the objects in your repository by performing Smalltalk full backups, using methods provided as part of the GemStone kernel. Smalltalk full backups are required if you want to reduce the number of extents in the repository or redistribute objects within the repository. During a Smalltalk full backup, dynamic internal data structures are not copied and will be rebuilt, which can, at least temporarily, improve the performance of such routine maintenance tasks as garbage collection.

In a Smalltalk full backup, the methods `Repository>>fullBackupTo:` or `fullBackupTo:MBytes:` save the most recently committed version of the repository in a way that is consistent from a transaction viewpoint. These methods force a checkpoint of the repository at the time the method is executed and then creates a backup from that checkpoint, copying all objects in the repository and arranging them in a compact form in one or more files.

You can make Smalltalk full backups while the repository is in use. Other sessions can continue to commit transactions, but those transactions are not included in the backup. Full backups require the GcLock, and so full backups cannot be made while other operations that hold the GcLock are running.

A Smalltalk full backup includes these three steps:

1. The Gem performing the backup scans the object table, building a list of objects to back up. This step runs in a transaction and can therefore cause a temporary commit record backlog in systems with high transaction rates. This step normally completes fairly quickly.
2. The Gem performing the backup next writes all shadow objects to the backup file. This step also runs in a transaction; furthermore, backing up shadow objects requires more disk I/O than backing up live objects, so the rate of objects backed up per second is slower in this step than in the next.

(For definitions of shadow and live objects, see “Basic Concepts” on page 255.)

3. In the final step, all remaining live objects are written to the backup file. This step is performed outside a transaction; if the Stone signals the session to abort, it will do so. This step takes the longest of the three.

The fullBackupTo: Methods

```
Repository>>fullBackupTo:filename
```

```
Repository>>fullBackupTo: arrayOfFileNames MBytes: mByteLimit
```

In these methods, *filename* or *arrayOfFileNames* specifies one or more files where the backup is to be created. You must specify the name of the files, not a directory name. You may include a relative or absolute path in addition to the file name.

If you use a relative path, the path is relative to the directory of the Gem process or linked session. For linked topaz sessions, this is the directory from which topaz was started. For RPC Gems, this is either specified by #dir: in the login parameters, or the home directory of the Gem’s UNIX user.

You can create backups on a remote node if the drive is NFS-mounted, or by using a network resource string (NRS) to specify the node name as part of the file name, and ensuring a NetLDI is running on the remote node. Backups cannot be made to a raw partition.

mByteLimit is either a single integer, or an array of integers with the same number of elements as *arrayOfFileNames*. This argument limits the maximum size of each file, except the last. If *mByteLimit* is one integer, each backup file will use that value; if it is an array of integers, each file will be limited by the matching entry. A value of 0 means the file sizes are unlimited.

In order to avoid running out of space for the backup, the last file is not limited, regardless of the size limit specified. If the number and size limit of *arrayOfFileNames* is too small to hold the entire backup, after each of the earlier files reaches its *mByteLimit*, the last file may grow significantly larger to contain the remainder of the backup.

WARNING

If there is not sufficient space to write the entire backup, the backup will return an error and deletes the incomplete backup files. Make sure you have sufficient disk space and the appropriate value for mByteLimit.

If you do not want to limit the size of the backup file, specify a *mByteLimit* of 0.

For example:

```
topaz 1> printit
"Create a full backup of the Repository"
SystemRepository
  fullBackupTo: {
    '/users/backups/August_20.1' .
    '/users/backups/August_20.2' .
    '/users/backups/August_20.3'
  }
  MBytes: 0.
%
true
```


This writes the backup into three files, named **August_20-1**, **August_20-2**, and **August_20-3**. Messages are written to the stone log indicating when the backup started and when it completed.

During the backup, after the initial period in transaction, the session is put into manual transaction mode so the backup won't interfere with ongoing garbage collection. When the backup completes, the session is left outside of a transaction. If you want to make changes to the repository after a backup, send `System beginTransaction` or `System transactionMode: #autoBegin`.

Backup fails to run or encounters an error

If the backup file already exists, a path cannot be found, or if any of the file names are empty strings, the method returns an error.

If multiple other sessions are performing repository scan operations, the system may not be able to acquire a GcLock. The backup will wait for up to 5 minutes for a lock to become available, otherwise it will fail and return an error. You can determine the session holding the GcLock by using:

```
System sessionIdHoldingGcLock
```

This method will return 0 if no session is holding the GcLock.

Backup (and restore) require at least one extra session be available, beyond the session that is starting the backup, and uses more sessions to write backups in system with multiple extents and to multiple backup files. If the number of users logged in is equal to the `STN_MAX_SESSIONS` setting, the backup will fail with an error.

If backup encounters an error, then any backup files that were created are automatically deleted.

Monitoring and Performance

The following performance and monitoring topics apply to full backups. The performance of on and offline extent snapshot backups depends on your operating system and disk performance and can be monitored and optimized outside of GemStone using OS level tools.

Shared Page Cache Size

You can often improve both backup and restore performance by increasing the size of the shared page cache.

Multi-threading

Full backups are written and restored multi-threaded to allow the reads and file writes to progress in parallel for faster performance.

The maximum number of threads is based on the number of extents in the repository and the number of backup files specified. For backup, threads may be deactivated and system impact reduced using the methods described in "Tuning Multi-Threaded Scan" on page 278. Restore is always done with maximum performance.

The multi-threaded algorithm uses one session per extent in your repository, up to a limit of 16 sessions. If there are not sufficient session slots available – if the number of users logged in is close to the `STN_MAX_SESSIONS` setting – then the backup or restore will use fewer sessions and performance will be slower. In this case, a message is printed to stdout (the topaz -l terminal) and to the stone log.

Cache Statistics

During the main part of a full backup, the statistic `ProgressCount` for the session performing the backup indicates the number of objects written to the backup file thus far. If you know the total number of objects in the repository, you can use this statistic to determine how far the backup has progressed.

Backups and Garbage Collection

NOTE

You will find it easier to understand the following discussion if you have first read and understood the section "Basic Concepts" on page 255.

Because shadow objects must be backed up, it is more efficient to run a Smalltalk full backup when there are few shadow objects. If possible, first check the statistic `PagesNeedReclaimSize`. If that statistic is high, run one or more Reclaim Gem sessions before performing the backup. (See "Admin and Reclaim Gems" on page 261.)

Dead objects waiting to be reclaimed (measured by the statistic `DeadNotReclaimedObjs`) are not backed up, as these objects are going to be deleted anyway.

Compressed Backups

It is possible to write and read full backup files in compressed mode. GemStone supports both `gzip` and `lz4` compression for backups.

Writing to, and reading from, a compressed file can be performed only to a local file system file or to a file system that is NFS-mounted.

Backup files written in compressed mode are automatically appended with the suffix `.gz` or `.lz4`, according to the selected compression.

All restore methods automatically detect whether a file is compressed or not and read the file accordingly. Even a backup originally created in uncompressed mode, then later compressed externally, is readable by `restoreFromBackup:.`

The following class methods in `Repository` are provided to create compressed full backups:

```
fullBackupGzCompressedTo: filename
```

```
fullBackupLz4CompressedTo: filename
```

These methods back up the receiver to a single backup file, which is written compressed in `gzip` or `lz4` format.

```
fullBackupGzCompressedTo: arrayOfFilenames mBytes: mByteLimit
```

```
fullBackupLz4CompressedTo: arrayOfFilenames mBytes: mByteLimit
```

These methods are similar to `fullBackupTo:mBytes:` except that the output file is written compressed in `gzip` or `lz4` format.

Verifying a Backup is Readable

To verify that a backup file is readable, use the GemStone utility `copydbf`. You can conserve disk space and reduce disk activity by specifying `/dev/null` as the destination. For instance:

```
% copydbf /users/backup/August_20-1 /dev/null
```

Checking Backup Start and Completion

The time a backup is started, and the time that it completes successfully, are written to the stone log. For multi-file backups, only the first filename is listed.

```
--- 08/25/17 14:55:59 PDT ---
    Full backup of the repository has been started.
      Host: ip6-localhost      ProcessId: 2930
      User: DataCurator      SessionId: 5
--- 08/25/17 14:57:23 PDT ---
    Full backup successfully completed by sessionId 5 to file:
bkup.dat
```

11.5 How to Restore from Backup

There are several circumstances under which you will want to restore from backup.

If you have disk errors or file corruption, or if you encounter object corruption in your repository, you will need to restore from backup and replay transaction logs to recover all work up to the time of the corruption.

Restoring from backup is also used to set up and refresh warm or hot standby systems, and to set up test environments that match production systems.

To make the repository smaller, or to redistribute objects among a different number of extents, or to change your system to use partial logging mode, you must restore from full backup. Restore from full backup may also improve space use and performance by recreating internal structures.

Note that if your intention is to redistribute objects over a different number of extents, if the number of extents during restore is the same as the number of extents when the backup was started, this takes precedence over the `DBF_ALLOCATION_MODE` configuration setting during restore. If the number of extents differs, then the `DBF_ALLOCATION_MODE` setting at the time of the restore controls the distribution of objects across extents.

The ability to restore from backup to recover from file or object corruption is critical to the reliability of your GemStone system. You should ensure that you regularly take backups, and from time to time, verify that the processes that you use to make the backups result in complete and usable backup files.

If you are concerned about losing work that is done between backups, ensure that you are in full transaction log mode. In this mode, the transaction logs record all commits in your repository and the transaction logs are not automatically deleted, so they can be replayed into a restored backup if they are needed.

There are two phases of restoring from backup:

Phase 1 - restore the backup. The process will vary depending on if you are restoring from an extent snapshot backup or from a full backup.

- ▶ To restore from extent snapshot backups, see “Restoring from an Extent Snapshot Backup” on page 220.
- ▶ For restore from a fullbackup, see “Restoring from a Full Backup” on page 222

Phase 2 - restore transaction logs. This phase is only possible in full transaction logging mode. If you are not in full transaction logging mode, any work done after the backup was made will be lost in the restored repository.

► “How to Restore Transaction Logs” on page 230

After the backup has been restored, the repository reflects its state at the time of the backup. All the objects are intact and ordinarily are clustered in a way similar to their organization in the original repository. This clustering reflects both explicit clustering of objects by the application and default clustering into the generic cluster bucket.

Restore Status

Before, during, and after restore from backup and from transaction logs, you can use the message `restoreStatus` to determine where you are in the process. This status is an attribute of the repository, not of the session, and persists across login sessions and stopping and restarting the Stone.

Not in restore mode

```
topaz 1> printit
SystemRepository restoreStatus
%
Restore is not active
```

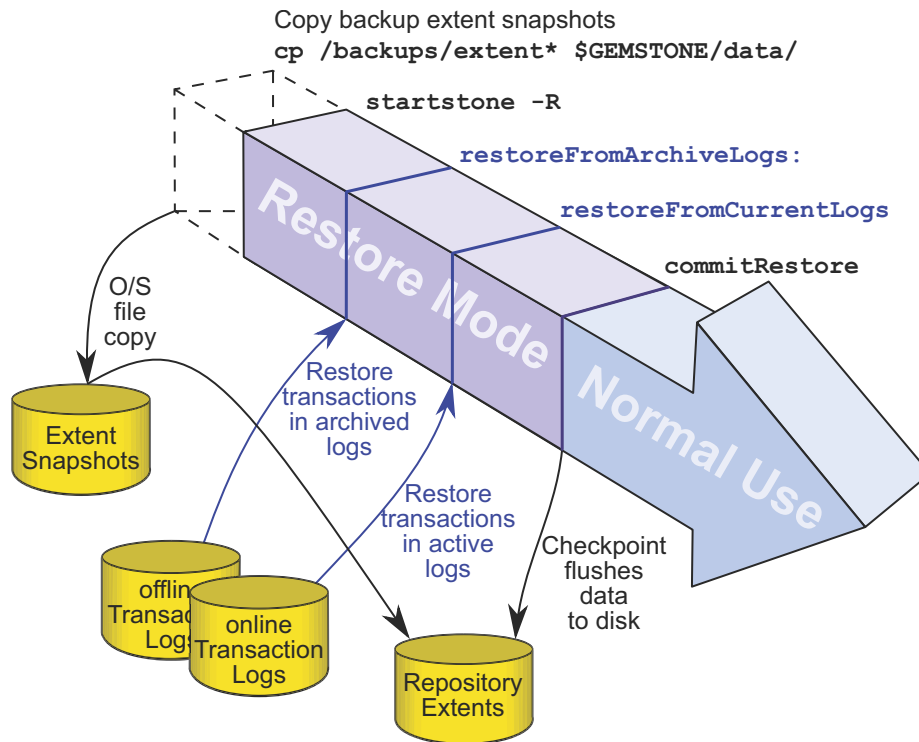
During restore from transaction logs

```
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from transaction log files, restored to 03/20/15
12:21:41 PDT, nextFileId = 1, record = 409 oldest fileId = 1
```

Restoring from an Extent Snapshot Backup

This section describes how to restore extent copy backups; that is, backups made using utilities such as `cp` to take a snapshot of the extent files. In order to recover, this backup must have been made while checkpoints were suspended for the entire time the copy was being made, or while the repository monitor was shut down; and snapshots of each extent must be available.

Figure 11.1 System Timeline: Restoring from a Extent Snapshot Backup



To restore your repository from a extent copy backup, use the following procedure:

Step 1. If GemStone is still running, tell all users to log out and use `stopstone` to stop the repository monitor.

Step 2. If you are restoring the repository because of a suspected GemStone failure, preserve a copy of the extents in case Technical Support wants to examine them.

Step 3. Delete all extent files specified by `DBF_EXTENT_NAMES` in your configuration file.

WARNING

Do NOT delete the transaction log files – leave them online in their current locations.

Step 4. Copy the operating system backup copies of the extent files to the locations specified by the `DBF_EXTENT_NAMES` configuration option.

Step 5. Ensure that there is space to create a log file. At least one of the directories specified by `STN_TRAN_LOG_DIRECTORIES` must have space available or one of the raw partitions must be empty. You may need to add entries to `STN_TRAN_LOG_DIRECTORIES` and `STN_TRAN_LOG_SIZES` in your configuration file.

Step 6. Start up the stone.

If full transaction logging (`STN_TRAN_FULL_LOGGING = True`) was in effect, start in restore mode to restore transaction logs. Use `startstone -R` to restart GemStone.

If partial transaction logging (`STN_TRAN_FULL_LOGGING = False`) was in effect at the time the backup was made, tranlogs are not restored. Restart Gemstone by invoking `startstone` in the usual manner. The restore process is now complete.

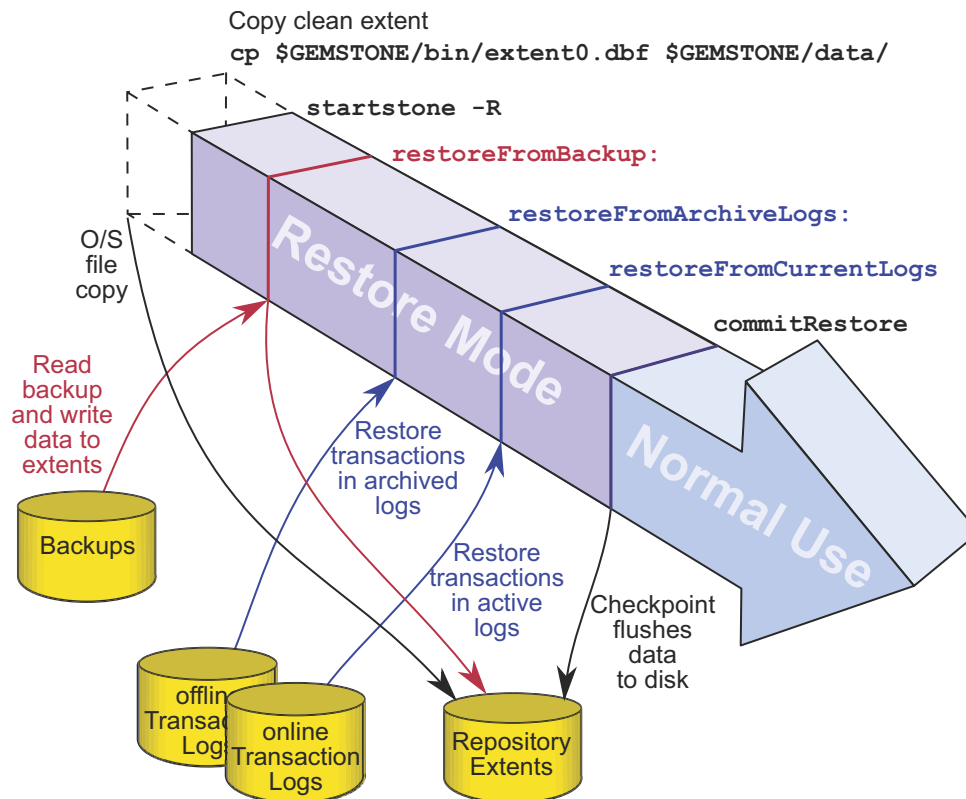
Step 7. If in full transaction logging, you can now restore transaction logs.

Continue with “How to Restore Transaction Logs” on page 230.

Restoring from a Full Backup

A GemStone full backup writes the contents of the repository into backup files. When restoring from a full backup, you start with a clean, empty extent, and restore the objects from the backup file into your repository.

Figure 11.2 System Timeline: Restoring from a Smalltalk Full Backup



To begin, you need a file copy (*not* a GemStone backup) of a good repository. We recommend that you use a copy of the `extent0.dbf` that was shipped in `$GEMSTONE/bin`, although any extent file that is a complete, uncorrupted repository will work. If you are using the backup/restore process to reduce the size of your extent, the new extent file must be smaller than your current extent.

Make sure that you have all backup files are complete. If the backup consists of multiple files, the complete set must be available.

The user restoring the backup must be the only user logged in to the server. The method that starts the restoration will suspend other logins.

NOTE

We recommend that you log in as DataCurator or SystemUser to restore the backup. If you start the restore as another user and that UserProfile disappears as a result of the restore, Topaz will see a fatal error.

To restore your repository from a Smalltalk full backup, use the following procedure.

Steps in restoring a full backup

Step 1. If GemStone is still running, tell all users to log out and use **stopstone** to stop the system.

Step 2. If you are restoring the repository because of a suspected GemStone failure, preserve a copy of the extents in case Technical Support wants to examine them.

Step 3. Delete all extent files specified in DBF_EXTENT_NAMES in your configuration file.

WARNING

Do NOT delete the transaction log files up to the time of the crash – leave them online in their current locations.

Step 4. Copy the clean, empty extent to the location of your primary extent, which is the extent listed first in DBF_EXTENT_NAMES.

Make sure there are no other extent files in that location. Do not copy any other extent files to the extent location. If you have more than one extent, the Stone repository monitor will create the new extents at startup.

Use **chmod** to give the clean extent copy the same permissions you ordinarily assign to your repository files.

For example:

```
% copydbf $GEMSTONE/bin/extent0.dbf $GEMSTONE/data/extent0.dbf
% chmod 600 $GEMSTONE/data/extent0.dbf
```

Step 5. Ensure that there is space to create a log file. At least one of the directories specified by STN_TRAN_LOG_DIRECTORIES must have space available or one of the raw partitions must be empty. You may need to add entries to STN_TRAN_LOG_DIRECTORIES and STN_TRAN_LOG_SIZES in your configuration file.

Step 6. Use **startstone -R** to restart the Stone.

The -R option starts the stone in restore mode and avoids creating an orphan transaction log.

For optimal performance, your extent files should be pre-grown during startup, rather than growing incrementally during restore. See “Pregrowing Extents to a Fixed Size” on page 45.

Step 7. Log in to GemStone as DataCurator or SystemUser using linked Topaz (**topaz -l**). Remember that the password will be the original one supplied when you installed GemStone, not necessarily the one you have been using.

NOTE

To perform the following steps, you must be the only user logged in to GemStone. Once you start the next step, other logins will be suspended.

Step 8. Restore the most recent full backup to the new repository by sending the message `restoreFromBackup:` or `restoreFromBackups:`. These methods automatically detect whether a backup is compressed or not and reads it accordingly.

```
topaz 1> printit
SystemRepository restoreFromBackup: 'backup.gz'
%
```

To restore from a multi-file backup, you must specify all the files in the backup, in the order the backups were created.

```
topaz 1> printit
SystemRepository restoreFromBackups:
  #( '/users/backups/August_20.1'
    '/users/backups/August_20.2'
    '/users/backups/August_20.3')
%
```

When restore from backup is complete, the session logs out.

```
[Info]: Logging out at 08/20/17 12:21:41 PDT
The restore from backup completed, with 97655 objects restored.
Ready for restore from transaction log(s).
```

If partial logging was in effect (`STN_TRAN_FULL_LOGGING = false`) at the time the backup was made, the final status line reads:

```
Restore complete. (Backup made while in partial logging mode.)
```

This status means that transaction logs cannot be restored. The repository is ready for ordinary use, and logins have been enabled.

If an error occurs during the restore, the system returns to the state it was in before restore. Determine the cause of the error and correct it, and return to Step 7.

Step 9. If full logging was in effect (`STN_TRAN_FULL_LOGGING = true`), the status line indicates the next step:

```
Ready for restore from transaction log(s).
```

If the version of `IcuLibraryVersion` in your restored backup is different than the version in the clean, empty extent, then you will need to exit the linked topaz environment.

Continue with “How to Restore Transaction Logs” on page 230.

Controlling Reclaim Activity When Restore Completes

During restore, pages that contain free space are by default not added to the scavengeable pages at the end of the restore. This avoids a load on the reclaim gems immediately after the `commitRestore`. In the normal course of operations as repository objects are operated on, pages with free space will tend to be reclaimed over time.

This can be controlled, so that pages with a specific percentage of free space are made scavangeable, so they will be reclaimed after the commitRestore. This will result in the largest amount of free space after this initial reclaim, at the expense of heavy reclaim load on the repository shortly after startup. Since reclaim requires pages, you should use some caution to avoid running out of free space before the newly reclaimed pages become available.

To explicitly specify the page free space percentage required to add pages to the scavangeable pages list, use the method `Repository >> restoreFromBackups: arrayOfFileNames scavangePagesWithPercentFree: aPercent`.

A *aPercent* value of 100 means no pages are added (the default), while 0 means pages with any free space at all are added.

For example,

```
topaz 1> printit
SystemRepository restoreFromBackups:
  #( '/users/backups/August_20.1'
    '/users/backups/August_20.2'
    '/users/backups/August_20.3')
scavangePagesWithPercentFree: 90
%
```

11.6 How to Make and Restore Secure Backup

Secure backups are similar to full backups, but include some important security features:

- ▶ They are digitally signed, ensuring the identity of the originator and that the backup file has not been modified. All backups created using the secure backup API are digitally signed.
- ▶ They may be encrypted, requiring a private key to restore. Encryption is optional, using AES-CTR with key sizes of 128, 192, or 256.

Secure backups require RSA keypairs, both for signing and for encryption (if used); DSA is not supported. For any secure backup, a signing keypair is needed; encrypted backups require additional separate keypair/s. The signing key may or may not require a passphrase, although a passphrase is recommended for security. The example keys included in the distribution require passphrases.

For an encrypted secure backup, you may use one or up to eight public keys, as well specifying the other details of the encryption. Restoring this backup later will require a private key that corresponds to any one of these public keys. This allows you to create a backup that can be restored by multiple other entities, without these entities having to share a single private key.

Secure backups, like fullBackups, can also be written to one or multiple files and may be uncompressed, compressed using zlib or compressed using lz4.

While secure and fullBackups are similar programmatic backups, the API methods are different. You cannot use the secure restore methods to restore a standard full backup, nor vice versa.

Creating a secure backup

Secure backups are created using the Repository method `secureFullBackupTo: . . .`, which allows you to specify filename or names, encryption type, and other details of the backup.

While making a secure backup requires more information to be passed in than an ordinary `fullBackup`, otherwise the process is similar. “How To Make a Smalltalk Full Backup” on page 215 describes the general process of making a full backup, including error conditions (other than certificate errors) and monitoring backups.

The method to create a secure backup is:

```
Repository >> secureFullBackupTo: arrayOfFileNames
  MBytes: mByteLimit
  compressKind: compressionKind
  bufSize: bufSize
  encryptKind: encryptionKind
  publicKeyCerts: anArrayOfString
  signatureHashKind: hashKind
  signingKey: signingKeyFn
  signingKeyPassphrase: aPassphrase
```

The following information must be provided:

arrayOfFileNames - a String containing a filename for the backup, or an array of strings for a multi-file backup. The extension `.sdbf` will be appended.

mByteLimit specifies the sizes of the backup files, as described under “How To Make a Smalltalk Full Backup” on page 215. 0 means unlimited; the last file of the backup is always unlimited size, to avoid inadvertent backup failures.

compressionKind - may be 0, 1, or 2. 0 indicates no compression, 1 specifies zlib/gzip compression, and 2 specifies lz4 compression.

bufSize - specifies the number of records to fit in a buffer. This would normally be 8 for uncompressed backups, 1 with gzip compression, and 16 with lz4 compression.

encryptionKind - may be 0, 1, 2, or 3. 0 indicates no encryption, and 1, 2, and 3 specify AES-CTR-128, AES-CTR-192, and AES-CTR-256, respectively.

anArrayOfString - an array of names of public certificate files, or nil if the backup will not be encrypted (if *encryptionKind* is 0). Up to 8 may be included. At least one of the private keys corresponding to these public keys will be needed in order to restore this backup.

hashKind - may be 1, 2, or 3, specifying SHA256, SHA384, or SHA512, respectively.

signingKeyFn - the name of the signing private key certificate file.

aPassphrase - the passphrase for the signing key certificate.

Setting the certificate directories

Before using the secure backup API to make or restore secure backups, you must set the keyfile directories (keyring), using the Gem configuration option `GEM_KEYRING_DIRS`. This can be set either in the configuration file used by the Gem process, or set at runtime.

The directories specified in GEM_KEYRING_DIRS are used to confirm there are matching public and private keyfiles for keys used by the secure backup operations.

The code examples below can be executed using the example certificates, by executing the following expression. Since this is a Gem runtime parameter, you will need to execute it each time after login.

```
System gemConfigurationAt: #GemKeyRingDirs
  put: {'$GEMSTONE/examples/openssl/certs' .
       '$GEMSTONE/examples/openssl/private' }.
```

Create an unencrypted secure backup

The following example code creates a secure backup in a single file, uncompressed and not encrypted, using the example key and passphrase in the GemStone distribution.

Example 11.2 Creating an unencrypted backup

```
SystemRepository
  secureFullBackupTo: '$GEMSTONE/data/backup1'
  MBytes: 0
  compressKind: 0
  bufSize: 8
  encryptKind: 0
  publicKeyCerts: nil
  signatureHashKind: 1
  signingKey: 'backup_sign_2_clientkey.pem'
  signingKeyPassphrase: (GsSecureSocket getPasswordFromFile:
    '$GEMSTONE/examples/openssl/private/backup_sign_2_client_pas
    swd.txt' )
```

Create an encrypted backup

The code in Example 11.3 creates an encrypted, compressed backup to three files:

Example 11.3 Creating an encrypted backup

```
SystemRepository
  secureFullBackupTo: { '$GEMSTONE/data/backup-1' .
    '$GEMSTONE/data/backup-2' .
    '$GEMSTONE/data/backup-3' }
  MBytes: 0
  compressKind: 2
  bufSize: 16
  encryptKind: 2
  publicKeyCerts: { 'backup_encrypt_1_clientcert.pem' .
    'backup_encrypt_2_clientcert.pem' }
  signatureHashKind: 1
  signingKey: 'backup_sign_2_clientkey.pem'
  signingKeyPassphrase: (GsSecureSocket getPasswordFromFile:
    '$GEMSTONE/examples/openssl/private/backup_sign_2_client_pas
    swd.txt' )
```

Restoring a secure backup

Restoring a secure backup is similar to restoring a fullBackup—the only difference is that a specialized method is used to pass in the additional required information. See the illustration on page 222.

For an encrypted secure backup, you must also specify one of the private keys corresponding to the public keys used in the encryption.

To restore a secure backup, following the steps under “Steps in restoring a full backup” on page 223, through step 7.

In Step 8, you will use one of the following secure backup restore methods:

```
Repository >> restoreFromSecureBackup: aFileName
  privateKey: aKey passphrase: aPassphrase
```

```
Repository >> restoreFromSecureBackup: aFileName
  scavengePagesWithPercentFree: aPercent
  privateKey: aKey passphrase: aPassphrase
```

```
Repository >> restoreFromSecureBackups: anArrayOfFileNames
  privateKey: aKey passphrase: aPassphrase
```

```
Repository >> restoreFromSecureBackups: anArrayOfFileNames
  scavengePagesWithPercentFree: aPercent
  privateKey: aKey passphrase: aPassphrase
```

These methods require that you have set the keyfile directories using the Gem configuration option GEM_KEYRING_DIRS. This can be set either in the configuration file used by the Gem process, or set at runtime. The directories specified in

GEM_KEYRING_DIRS are searched for the public key corresponding to the private signing key, and a private key that corresponds to a public encryption key.

Restore an unencrypted backup

The following example restores a single file secure backup that was not encrypted. #GemKeyRingDirs must include a directory containing the public key for the signing certificate, which is used to verify the backup. This public key can be extracted from the backup using the **copydbf -V** command, so anyone can restore this backup; it is secured against modification, but not private.

Example 11.4 Restore an unencrypted secure backup

```
SystemRepository
  restoreFromSecureBackup: 'backup.sdbf'
  privateDecryptionKey: nil
  passphrase: nil.
```

Restore an encrypted backup

The code in Example 11.5 restores a encrypted backup in three files. The private key backup_encrypt_1_clientkey.pem and passphrase used for decryption corresponds to the public key backup_encrypt_1_clientcert.pem which was used to make the backup. The private key is required in order to be able to restore the encrypted backup.

While copydbf -W will extract the public keys from the backup, these are not sufficient for restore.

GemStone automatically detects the encryption type and compression type.

Example 11.5 Restore an encrypted backup

```
SystemRepository restoreFromSecureBackups: {
  '$GEMSTONE/data/backup-1' .
  '$GEMSTONE/data/backup-2' .
  '$GEMSTONE/data/backup-3' }
  privateDecryptionKey: 'backup_encrypt_1_clientkey.pem'
  passphrase: (GsSecureSocket getPasswordFromFile:
  '$GEMSTONE/examples/openssl/private/backup_encrypt_1_client_passwd
.txt' )
```

After restoring the secure backup, you can continue with restoring transaction logs or performing the `commitRestore`, as described in “How to Restore Transaction Logs” on page 230.

Verifying the digital signature

Secure backups are always digitally signed. You can use `copydbf`, or the GemStone script `verify_backup_with_openssl` to verify the signature. This script uses publicly

available openssl (by default the executable `$GEMSTONE/bin/openssl`) to perform the verification.

To verify the digital signature, you will need the public certificate file corresponding to the private key that was used to sign the backup. Since the public key is included in the backup, you can perform the verification without specifying the certificate, or by extracting the certificate using the `copydbf -X` option, but this not verification in a secure sense.

```
copydbf -v backupFile -K certificatesDirectory
```

Use `copydbf` to verify the backup file `backupFile`, with the certificates located in the directory `certificatesDirectory`.

```
verify_backup_with_openssl.sh backupFile certFile
```

Verify the secure backup file `backupFile` using certificate `certFile`

The following examples verify the digital signature in the unencrypted backup in Example 11.2:

Example 11.6 Verify digital signature

```
% copydbf -V backup1.sdbf -K $GEMSTONE/examples/openssl/certs/
Source file: backup1.sdbf
  File type: secure backup  fileId: 0
in a backup set with 1 files
  ByteOrder: Intel (LSB first)  compatibilityLevel: 850
  The file was created at: 09/07/2017 23:26:16 PDT
[Info]: Digital signature successfully verified.

% verify_backup_with_openssl.sh backup1.sdbf
  $GEMSTONE/examples/openssl/certs/backup_sign_2_clientcert.pem
[Info]: Using certificate file
  /lark1/GS/examples/openssl/certs/backup_sign_2_clientcert.pem
[Info]: Digest kind is SHA-256
[Info]: Invoking openssl to perform the digital signature
  verification:

Verified OK
```

11.7 How to Restore Transaction Logs

The second phase of restoring the repository is to roll forward from the state at the starting point of the last backup to the state of the last committed transaction. This action repeats the transactions in the order in which they were committed.

You can do this only if the `STN_TRAN_FULL_LOGGING` configuration option was set to `True` at the time the backup was made. You cannot restore transaction logs that are not part of a sequence of tranlogs that includes the backup. Since restore breaks this sequence, the transactions being restored cannot span a more recent restore.

Note that while backup files can be written in either uncompressed or compressed format, transaction logs are always written in uncompressed format. However, transaction logs may be compressed, using copydbf, gzip, or lz4, before archiving them. These compressed tranlogs can be restored directly, without having to manually uncompress them.

At this point, GemStone should be running and in restore mode, following a restore from either an extent snapshot backup or from a full backup. The following steps describe the most common case of restoring the transaction logs.

CAUTION

Ordinarily, you will restore transactions from all log files written since the backup. If for some reason you plan to omit one or more log files, refer to the section "Special Cases and Errors in Restore" on page 232.

Step 1. Log in to GemStone as DataCurator or SystemUser using linked Topaz (**topaz -l**).

Step 2. Determine which transaction logs are needed for restore and their locations. The method `restoreStatus` identifies the earliest transaction log that is needed. In this example it is `tranlog6.dbf`:

```
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
  restored to 08/02/17 13:26:31 PST
  next fileId = 6, record = 9.
```

Compare the `fileId` in the message with the names of the transaction log files in the directories specified in `STN_TRAN_LOG_DIRECTORIES`. For transaction logs in the file system, `fileId` forms the numeric portion of the file name, `tranlogNN.dbf`. For transaction logs in raw partitions, use **copydbf -i** *fileName* to display the `fileId`.

Transaction log files that are located in a directory specified in `STN_TRAN_LOG_DIRECTORIES` are "current". If some required transaction logs have been moved to another location, they are "archive" logs, and are restored using a different method.

Step 3. Restore archive transaction logs, if any.

If any of the tranlogs to be restored are not in one of the current tranlog directories, collect the names of directories containing all these archive logs, and restore using `Repository>>restoreFromArchiveLogs:` or related methods.

You will have to login prior to running this step.

```
topaz 1> printit
SystemRepository restoreFromArchiveLogs:
  #( 'GS-archive' )
%
```

See the method comments in the image for details. A directory location can include an NRS for a remote node, but a NetLDI must be running on that node.

If you encounter a failure because of a truncated or corrupted transaction log, refer to "Errors While Restoring Transaction Logs" on page 236.

Step 4. Before continuing to restore tranlogs, you must log in again. Restore operations terminate the session when complete.)

Step 5. Restore transactions from the current log files by executing the method `Repository>>restoreFromCurrentLogs`. All the remaining log files must be in directories or raw partitions specified in `STN_TRAN_LOG_DIRECTORIES`.

```
topaz 1> printit
SystemRepository restoreFromCurrentLogs
%
Restore from transaction log(s) succeeded.
```

Step 6. If restoration from the transaction logs was successful, send the message `commitRestore` to tell the system that you are finished restoring. After this, no further logs can be restored, and normal user commits will be allowed.

You will have to login again prior to running this step.

```
topaz 1> printit
SystemRepository commitRestore
%
Restore from transaction log(s) succeeded. commitRestore
succeeded
```

If you send `commitRestore` prior to `restoreFromCurrentLogs`, a warning is issued because all previously committed transactions may not have been restored. However, this usage provides a way to recover as much as is available when a log file has been corrupted or lost.

Step 7. Make a new GemStone backup as soon as operational circumstances permit.

11.8 Special Cases and Errors in Restore

If all transaction logs needed to restore up to the current time are available, transaction log restore is simple. However, in some cases transaction logs may be missing or corrupt, or you may wish to restore to an earlier point in time. This section describes these special cases and problems that you may encounter during transaction log restore.

Precautions When Restoring a Subset of Transaction Logs

When you determine the need restore an incomplete set of transaction logs, be aware of the likely consequences:

- ▶ Obviously, the omitted transactions will be lost. Presumably that is unavoidable or intentional.
- ▶ Less obviously, it may be impossible to reverse your action later and restore the omitted logs. Operations after the first `commitRestore` create a time fork in the repository, and attempting to reverse the course later results in inconsistent data and object audit errors. For a detailed example illustrating this, see the following discussion on Fork-in-Time Scenario.

If there is any chance that you may want to restore from the omitted transaction logs later, prior to restore archive the repository backup and all transaction logs required for complete restore to a separate location. The transaction logs should not be on any directory listed in `STN_TRAN_LOG_DIRECTORIES`.

Later, if you wish to perform a second restore, you can repeat the entire restore process, including restoring any omitted transaction logs.

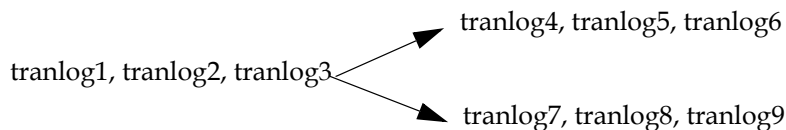
Since any new work done in the partially restored system constitutes a “Fork-in-Time”, the work done after the partially restored system’s `commitRestore` cannot be restored to this second restored system. That work will be lost.

Fork-in-Time Scenario

In some cases, you may encounter problems with restoring from your most recent backup file and must restore from an earlier backup. This scenario presents a risk of transaction logs that are out of sequence due to a “fork-in-time.” Consider the following sequence of repository events:

1. Generate backup1.
2. Generate transaction logs `tranlog1`, `tranlog2`, `tranlog3`.
3. Generate backup2.
4. Generate transaction logs `tranlog4`, `tranlog5`, `tranlog6`.
5. Restore backup2.
6. `commitRestore` (without replaying transaction logs `tranlog4`, `tranlog5`, `tranlog6`).
The repository is now at same state as Step 3.
7. Generate transaction logs `tranlog7`, `tranlog8`, `tranlog9`.
8. Restore backup1.
9. Replay transaction logs `tranlog1` through `tranlog9`.

In terms of the repository lifecycle, this scenario has two timelines, with a fork-in-time at the end of `tranlog3`:



If, at step 5, we also restored the transaction logs (`tranlog4`, `tranlog5`, `tranlog6`), the resulting sequence could be replayed without problems. The problem is caused when the continuity of the transaction log chain is broken.

After restoring backup1 in step 8, it would be possible to safely replay transaction logs `tranlog1` through `tranlog6` without problems, but any changes made in (`tranlog7`, `tranlog8`, `tranlog9`) would be lost.

During step 9, the replay of (`tranlog7`, `tranlog8`, `tranlog9`) is likely to produce problems. If any object changes made in (`tranlog4`, `tranlog5`, `tranlog6`) are logically inconsistent with those made in (`tranlog7`, `tranlog8`, `tranlog9`), possible errors are wide-ranging, including `UTL_ASSERT/UTL_GUARANTEE` errors or errors of the form:

```
recovery/restore: invalid operation XXXXXXXXXXXX
Transaction expected to abort.
non-empty invalidObjs in recover.c:commitTran
```

In the worst case, errors may not be written to the Stone log during transaction log replay, but the final repository may be corrupted in obscure ways. If the corruption is structural,

it may be detected by an object audit (described on page 134). Otherwise, the corruption may go undetected unless picked up by application code.

If you are presented with a situation wherein you are forced to restore from an earlier backup, keep in mind the following:

1. Be aware of the fork-in-time phenomenon and avoid restore/replay operations that would create a fork.
2. When restoring into an ongoing transaction log sequence, only restore a backup file generated earlier within that same sequence, and then replay *all* transaction logs in that sequence generated since that backup.
3. If for some reason you cannot follow guideline 2, realize that you cannot restore from an earlier backup and replay transaction logs beyond the point of the initially restored backup.

Restoring Logs up to a Specific Log

To restore transaction logs, stopping at a specific log, execute `Repository>>restoreToEndOfLog:fileId`. This restores all transaction logs up to and including the specified transaction log. All tranlogs from the next tranlog required through the specified tranlog must be available. For example:

```
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
restored to 08/02/17 13:51:19 PST
next fileId = 7, record = 0 oldest fileId = 7
topaz 1> printit
SystemRepository restoreToEndOfLog: 15
%
[Info]: Logging out at 08/24/17 14:37:07 PDT
Restore from transaction log(s) succeeded.
```

If the transaction logs to be restored are in an archive location, use the similar methods `restoreFromArchiveLogs:toEndOfLog:` or `restoreFromArchiveLogs:toEndOfLog:withPrefix:.`

```
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files, restored to 08/02/17
13:51:19 PST, next fileId = 7, record = 0 oldest fileId = 7
topaz 1> printit
SystemRepository
    restorefromArchiveLogs: #(GS-archive)
    toEndOfLog: 15
%
[Info]: Logging out at 08/24/17 14:37:07 PDT
Restore from transaction log(s) succeeded.
```

Restoring Logs to a Point in Time

Ordinarily, the methods to restore one or more transaction logs restores each individual transaction within the log file. However, you can specify an earlier stopping point and restore only part of a transaction log, by sending one of the following messages:

```
restoreToPointInTime: aDateTime
restoreFromArchiveLogs: arrayOfDirSpec toPointInTime: aDateTime
restoreFromArchiveLogs: arrayOfDirSpec toPointInTime: aDateTime
    withPrefix: tranlogPrefix
```

Restoration will stop at the first repository checkpoint that originally occurred at or after *aDateTime*. This may be several minutes after *aDateTime*, depending on the checkpoint frequency in the transaction log.

To display the time a transaction log was started and the time of each checkpoint recorded in it, use **copydbf -I *fileName***. By default, the interval between checkpoints is five minutes. For example:

```
% copydbf -I tranlog2.dbf
```

```
Source file: tranlog2.dbf
file type: tranlog  fileId: 2
byteOrder: Sparc (MSB first) compatibilityLevel: 933
The file was created at:      08/25/17 14:55:59 PDT.
The previous file last recordId is 69.
Scanning file to find last checkpoint...
Checkpoint 1 started at: 08/25/17 14:55:59 PDT.
    oldest transaction references fileId -1 ( this file ).
Checkpoint 2 started at: 08/25/17 14:57:23 PDT.
    oldest transaction references fileId -1 ( this file ).
File size is 2.2 MBytes (4350 records).
```

The method to use to restore to a point in time depends on if the logs are archive (not in a directory on STN_TRAN_LOG_DIRECTORIES), OR ONLINE (IN A DIRECTORY USED FOR CURRENT TRANSACTION LOGS).

If the point in time that you wish to restore to occurs in an current/online transaction log, first restore any archives logs using `restoreFromArchiveLogs:`.

Then, restore all current logs up to a specified time. The following example restores the repository to the first checkpoint that would have included a commit on August 22, 2017 at 2:56:00 p.m.:

```
topaz 1> printit
SystemRepository restoreToPointInTime:
    (DateTime fromString: '22/08/2017 14:56:00').
%
```

To restore to a point in time that is in an archived tranlog, use the method `restoreFromArchiveLogs:toPointInTime:` or `restoreFromArchiveLogs:toPointInTime:withPrefix:`. This second method allows you to also specify alternate file prefixes, if you rename files as part of the archive process.

The following sequence restores the repository to the first checkpoint that would have included a commit on August 22, 2017 at 2:56:00 p.m.:

```
topaz 1> printit
SystemRepository restoreFromArchiveLogs:
    #( 'GS-archive' )
    toPointInTime:
        (DateTime fromString: '22/08/2017 14:56:00').
%
```

You can continue restoring past *aDateTime* by issuing further restore messages.

Errors While Restoring Transaction Logs

Missing Transaction Log File

If a transaction log file in the sequence is missing, the tranlog restore stops at that point, and reports an error if it detects the existence of later transaction logs.

For example, if you have `tranlog1.dbf` through `tranlog10.dbf`, but `tranlog4.dbf` is missing, `restoreFromCurrentLogs` stops after restoring from `tranlog3.dbf`.

```
topaz 1> printit
SystemRepository restoreFromCurrentLogs.
%
[Info]: Logging out at 08/02/17 13:26:31 PST
ERROR 4049 , Restore from transaction log failed,
EndOfAllLogs reached after fileid 3 before last log for
recovery 10 found.
```

The tranlog after the one reported in the error is the one that is missing. You can also execute the method `restoreStatus` to identify the next log file explicitly. Locate the missing file or files, and then continue the restore process.

Truncated or Corrupt Transaction Log File

If a transaction log is truncated or corrupt, it may not be noticed until the next transaction log is restored. This may occur, for example, if you have an undetected disk full condition when copying a transaction log.

The truncated log may restore successfully, but when the next log is restored, the gap is detected and the error is reported.

In the following example, `tranlog6.dbf` is truncated, and `restoreFromCurrentLogs` reports an error.

```
topaz 1> printit
SystemRepository restoreFromCurrentLogs
%
[Info]: Logging out at 08/24/17 14:37:07 PDT
ERROR 4049 , Restore from transaction log failed
Log with fileId 6 is truncated or corrupt, or log 7 is
corrupt.
```

Logging in again and checking the restore status confirms that `tranlog6.dbf` is incomplete:

```
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files, restored to 08/02/17
13:51:19 PST, next fileId = 6, record = 4409 oldest fileId = 6
```

After locating a complete, uncorrupted copy of `tranlog6.dbf`, it is copied into the appropriate directory and the restore is done again:

```
topaz 1> printit
SystemRepository restoreFromCurrentLogs
%
[Info]: Logging out at 08/24/17 14:37:07 PDT
Restore from transaction log(s) succeeded.
```

You can verify that this and any later transaction logs were restored by logging in again and checking the restore status:

```
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files, restored to 08/02/17
13:51:19 PST, next fileId = 11, record = 4409 oldest fileId =
11
```

Since in this case all available transaction logs are now successfully restored, login again and commit the restored repository:

```
topaz 1> printit
SystemRepository commitRestore
%
Restore from transaction log(s) succeeded., commitRestore
succeeded
```

If you cannot find an undamaged copy of the transaction log, you cannot restore any further logs. Executing `commitRestore` will commit as much as has been restored. However, if there is any chance of a finding a good copy, see the discussion, "Precautions When Restoring a Subset of Transaction Logs".

Recovering from File System Problems

We recommend disk or operating system mirroring for applications that cannot tolerate the risk of data loss. In particular, recent transaction logs should be mirrored, or at minimum copied to an archive location on a frequent basis. In the case of a disk failure or a corrupt file system, if any of the transaction logs created since the last backup are corrupt or unusable, this recent work may be permanently lost.

In the case of disk failure or a corrupt file system, the file system must be repaired or restored. The most reliable strategy is to restore GemStone from backup, restoring copies of all transaction logs for which you have uncorrupted copies.

However, if you have important work that may be lost, you may want to attempt recovery of the existing repository. If each of these steps completes successfully, your repository is uncorrupted and you can resume normal operations.

Step 1. Perform page audit

Execute `pageaudit` per the instructions under “Page Audit” on page 132, to verify page-level integrity of the repository.

Step 2. Restart GemStone

Step 3. Perform object audit

Execute `objectAudit` per the instructions under “Object Audit and Repair” on page 134 to verify objects in the repository. This may take some time.

Some types of `objectAudit` failures indicate corruption in internal GemStone structures, which are rebuilt during restore of a full backup. If `objectAudit` reports errors, it may be worthwhile to attempt to make a `fullBackup` of the repository. If this succeeds, restoring it may provide a uncorrupted repository.

Warm and Hot Standbys

For high-availability production systems, it is a serious problem if the repository has an unexpected error and has to be shut down, or possibly require restore from backup. While such problems are rare, critical systems must be prepared.

For such systems, a second GemStone system can be kept running in parallel, so it can be brought into use with minimal downtime. GemStone provides several options for standby systems.

Warm Standby (page 240)

describes a standby system in which entire transaction logs may be manually transmitted and restored into the standby system.

Hot Standby (page 242)

describes how to setup processes to automatically transmit transaction log records as they are generated, and automatically restore the records into the standby

12.1 Overview

Customers with critical, high-availability systems may want to keep a duplicate of a production GemStone server running almost in parallel as a standby system. This duplicate continually runs in restore mode, restoring transactions from the production server. If anything goes wrong with the primary production server, the standby can be brought into use very quickly.

The production system is referred to here as the primary or master system. The standby system is also referred to as the slave system.

Following a failover, these roles change; the standby system becomes the master and users log in to perform work. Often the system that was previously the master, after correcting the problem that caused failover, is updated to become the new standby.

To operate a standby, the primary system must be running in full logging mode, as described under “Logging Mode” on page 49.

This chapter discuss how to set up the standby server and the process for restoring logs, which differ between warm and hot standbys. For general information about restoring

backups and transaction logs, see “How to Restore from Backup” on page 219. This discussion assumes you are familiar with that procedure.

12.2 Warm Standby

With a warm standby, transaction logs from the primary system are manually copied to the standby system as each log is completed. The standby runs in restore mode, and restores each log as it is closed.

When failover is needed, the final transaction log from the primary is copied over and restored, and the standby system performs a `commitRestore` and is then available for use as the new primary server.

An important point to remember is that the transaction logs copied from the production server, called the *archive logs* here, must be kept separate from transaction logs created by the duplicate server. You can do that by using different log directories or different file name prefixes.

Setup and run the warm standby

Step 1. Install the duplicate server. It is best to do a complete GemStone installation on a second node.

Step 2. Decide on a naming convention or location that you will use on the duplicate server to keep the archive logs separate from those being created by the duplicate server itself. For instance, if both Stones use the default prefix of *tranlog*, you might copy `tranlog123.dbf` on the production server to `$(GEMSTONE)/data/prodtranlog123.dbf` on the duplicate server.

Step 3. Make an extent copy backup, or a full backup of the primary system. You’ll have to do this at least once, when you start this system; however, regular backups will simplify matters when you need to synchronize the primary and the standby systems.

Step 4. Copy the extent backups, or restore the full backup, into the duplicate server. If you use extent copies, use the `-R` option to start the Stone, which causes the Stone to enter restore mode. For instance, **startstone -R**.

Step 5. As each transaction log completes on the primary system, move the log to a file system accessible to the warm standby GemStone installation. Avoid moving these logs into the transaction log directory that the warm standby uses for its own transaction logs.

Wait several seconds after the new log is created before copying the old log, to ensure the completion of any asynchronous writes.

You can limit each transaction log to a tolerable amount of information either by limiting transaction log size or by starting a new log at regular time intervals:

- ▶ *Size-based:* Limit the transaction log size, as described under “Choosing the Log Location and Size Limit” on page 51. When a transaction log grows to the specified limit, GemStone starts a new transaction log.
- ▶ *Time-based:* On your primary system, run a script at regular intervals that terminates the current transaction log and starts a new one, using the method `System class >> startNewLog`.

Step 6. On the warm standby, restore the transaction logs as they are available.

Since the transaction logs for the primary are not in the STN_TRAN_LOG_DIRECTORIES of the standby, you will restore from an archive log directory for these tranlogs. You can restore from an archive log directory using one of the following methods:

```
Repository>>restoreFromArchiveLogs:
Repository>>restoreFromArchiveLogs:tranlogPrefix:
Repository>>restoreFromArchiveLogs:toEndOfLog:
Repository>>restoreFromArchiveLogs:toEndOfLog:tranlogPrefix:
```

The tranlogPrefix: argument allows you to use a different setting for STN_TRAN_LOG_PREFIX on the production and standby systems.

Since restoring transaction logs terminates the session, you will need to login for each restore. For example:

```
topaz> login
<details omitted>
successful login
topaz 1> printit
System restoreFromArchiveLogs: {'GS-archive'}.
%
```

Step 7. Repeat Step 5 and Step 6 as necessary.

You may find it necessary to shut down the standby from time to time. Ensure that you shut down the stone using stopstone. This does not affect the restore status.

Note that if the standby is not shut down cleanly (i.e. an unexpected shutdown), the restarted system has the status of its last checkpoint. You may need to restore tranlogs again that were previously restored.

Activate the warm standby in case of failure in the primary

Step 1. If the primary system fails, replay its latest transaction log on the standby system.

Step 2. On the standby server, send the message `Repository>>commitRestore` to terminate the restore process and enable logins.

Step 3. Client applications will have to reconnect to the standby system, which now becomes the primary system. Applications may have to perform their own failure recovery code as necessary, as well.

NOTE

Design your application and configuration so that, after a failure occurs and the standby is activated, client applications can reconnect to the new primary correctly.

Step 4. Correct the problem on the failed system and restart it.

Depending on how much time has elapsed since the standby system became the primary system, either make a full backup of the new primary system and restore it on the system that failed, or replay the new primary system's transaction logs on the system that failed. Maintain that system in restore mode as the new standby.

12.3 Hot Standby

A Hot Standby provides faster failover since it can always remain synchronized with the primary server. If failover is needed, all that is required is to stop tranlog transmittal and restore, perform the final commitRestore, and the repository is ready for use.

As with the warm standby, a backup of the primary (or master) repository is installed on a standby (or slave) system. Then, special processes run to transmit the transaction log records as they are generated on the primary system. Individual transaction records, rather than entire transaction logs, are transmitted in compressed form between the systems. The standby system runs in a special mode where it will continuously restore the transaction log records. As a result, the standby system can keep closely synchronized with the primary.

Precautions regarding tranlog sequences

Since the hotstandby process relies on a logical sequence of tranlogs, some care must be taken to avoid situations such as described under “Fork-in-Time Scenario” on page 233.

For example, if you restore from backup on the primary, you must make a new backup of the primary, and restore this into the hot standby. Likewise, you cannot restore a backup into both the primary and the standby, since this creates a fork in the logical sequence of tranlogs, regardless of the tranlog numbering. While in some cases the automated process that transmit and restore the transaction log records will detect this, there may be cases where the processes will appear to hang waiting for the correct logs.

Hot standby processes

logsender

The **logsender** process runs on the master system. It determines when new transaction records are available on the primary system, and sends these records in compressed form to the standby slave system.

The logsender process is started using the **startlogsender** utility command. When starting the logsender, you must specify:

- ▶ The address and port to listen on for connections from a logreceiver on a slave system.
- ▶ all directories or raw partitions containing transaction logs generated by the master system.
- ▶ the name of the master stone. While technically optional, without this the logsender will not be notified of new data to send.

There are other optional arguments. See **startlogsender** on page 351 for more specific information on the arguments for this command.

A logsender continues to run when the associated master stone is shut down, and reconnects to the stone when the stone is restarted. Since it is continuously listening for connection requests from a logreceiver, the connection with the logreceiver can also be automatically reestablished after a disconnect, provided the logreceiver is running.

The logsender must be stopped explicitly using the **stoplogsender** utility command. See **stoplogsender** on page 361 for more details.

logreceiver

The **logreceiver** process runs on the standby slave system. It receives transaction logs from the master system and writes them to a location where the slave stone can restore them.

The logreceiver process is started using the startlogreceiver utility command. When starting the logreceiver, you must specify:

- ▶ The address and port that the logsender on the master system is listening on.
- ▶ One or more directories to write incoming transaction logs from the master system. It is not recommended to use raw partitions.
- ▶ the name of the slave stone. While technically optional, without this the logreceiver cannot notify the slave stone that new data has arrived.

There are other optional arguments. See **startlogreceiver** on page 349 for more specific information on the arguments for this command.

A logreceiver continues to run when the associated slave stone is shut down, and reconnects to the stone when the stone is restarted. If the connection to the logsender is lost, the logreceiver will attempt to reconnect.

The logreceiver must be stopped explicitly using the stoplogreceiver utility command. See **stoplogreceiver** on page 360 for more details on this command.

Continuous Restore Mode

A slave system in a hot standby runs in continuous restore mode. In this mode, it can restore individual transaction records as they become available.

To enter continuous restore mode, execute:

```
SystemRepository continuousRestoreFromArchiveLogs:  
anArrayOfRestoreDirectories
```

To exit continuous restore mode, execute:

```
SystemRepository stopContinuousRestore
```

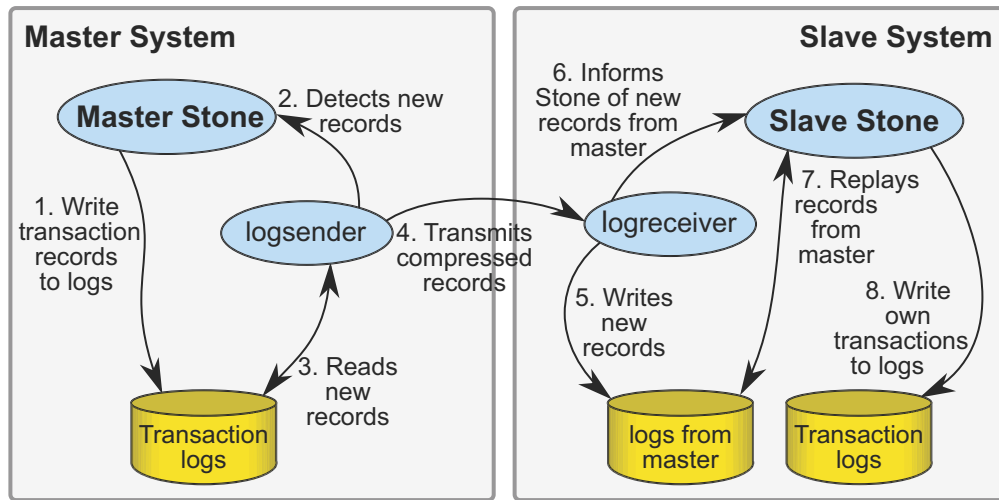
You must be in restore mode before you can enter continuous restore mode, and executing stopContinuousRestore leaves the stone still in restore mode.

You may not execute the commitRestore, which exits from restore mode, without first exiting continuous restore mode.

Transaction Record Transmittal

The process that transaction records follow from the master to the slave is described in this section.

Figure 12.1 Processes in a hot standby



As transactions are committed on the master stone, transaction log records are written to the transaction logs – this occurs for all GemStone systems. The logsender process is logged into the master stone and is aware when new transaction records are generated. The logsender also has an established connection with the logreceiver process. The logsender transmits the transaction log records in compressed form to the logreceiver.

The logreceiver accepts the transaction log records and writes them to the slave system's directories for restore logs. The logreceiver is logged into the slave stone, and makes the slave stone aware that new transaction records are available for restore.

The slave stone is running in continuous restore mode, and restores the transaction records.

When a logsender or logreceiver is logged into a stone, or while a stone is in continuous restore mode, the stone cannot restore a full backup or restore tranlogs (using other restore methods), nor can it perform a commitRestore.

Multiple standby repositories

A given master stone may have multiple slave stones. Each slave system will have a separate logreceiver process running. The logsender on the master system can transmit data to up to 5 logreceiver processes.

To setup and run the hot standby

- Step 1.** Install the slave server. It is best to do a complete GemStone installation on a second node.
- Step 2.** Decide on a directory location that you will use on the slave server to keep for the logs transmitted from the master. This should be a separate directory or directories from the tranlog directories of the slave stone.
- Step 3.** Make an extent copy backup, or a programmatic full backup of the primary system. This must be a backup from the primary system; you cannot restore a backup into both the primary and standby, since restore creates a fork-in-time (see page 233).

You'll have to do this at least once, when you start this system, and after each restore from backup or upgrade on the primary system. Regular backups will simplify matters when you need to synchronize the primary and the standby systems.

Step 4. Copy the extent backups to the slave system and start the stone using the `startstone -R -N` options; or start the stone on a clean extent and restore the full backup. The slave stone must be running in restore mode.

Step 5. Start the logsender process on the master system using the `startlogsender` utility.

Before starting the logsender, you will need to determine the set of directories that contain transaction logs on the master system. This will include all the entries in the master stone's `STN_TRAN_LOG_DIRECTORIES`. If the master stone's transaction logs are copied to another directory as part of an archive process, these archive directories may also need to be specified.

You will also need to select an port number that is unused on the master and slave systems for the logsender to listen on.

For example:

```
startlogsender -P 57222 -A masterListeningAddress -T $GEMSTONE/data
-s masterStone.
```

Step 6. Start the logreceiver process on the slave system using the `startlogreceiver` utility.

You will use the same port as the logsender, and the directory or directories you determined in Step 2

```
startlogreceiver -P 57222 -A masterNode
-T /gemstone/masterTransLogs -s slaveStone.
```

Step 7. Put the slave stone into continuous restore mode. To do this, log into the slave system and execute `Repository >> continuousRestoreFromArchiveLogs:`, passing in the list of directories you determined in Step 2. After entering continuous restore mode, this method will exit and you can log out.

Activate the hot standby in case of failure in the primary

In case of a failure in the master system, perform the following steps:

Step 1. Confirm that all tranlog records have been transmitted to the slave system, and that the slave system has restored all transaction log records.

To do this, provided the master system is operational, check the results of `restoreStatus` on the slave system, and compare this to the results of `copydbf -i` on the final tranlog on the master system.

Step 2. If the master system is operational, stop the logsender process on the master system using the `stoplogsender` utility command.

```
stoplogsender -P 57222
```

Step 3. On the slave server, stop the logreceiver process using the `stoplogreceiver` utility command.

```
stoplogreceiver -P 57222
```

Step 4. On the slave system, send the message `Repository>>stopContinuousRestore` to exit continuous restore mode, then

`Repository>>commitRestore` to terminate the restore process and enable logins. The slave system is now ready for use.

Step 5. Client applications will have to reconnect to the slave system, which now becomes the primary system. Applications may have to perform their own failure recovery code as necessary, as well.

NOTE

Design your client applications so that, after detecting a failure, they can determine which system is the new primary and reconnect correctly.

Step 6. Correct the problem on the failed former-master system and restart this as the new slave system.

You will need to make a fresh backup of the new master system and restore this into the new slave (the former master), prior to setting up the hot standby again.

Planned failovers

When you have a planned failover, there are additional steps you can take to ensure that all records have been transmitted, and stop new work from taking place on the master during the failover itself.

Step 1. On the master system, execute:

```
SystemRepository suspendCommitsForFailover
```

which will suspend commits and perform a checkpoint. This is the failover timestamp. At this point, no changes can be made in the master that will affect the transaction logs: commits are suspended, including garbage collection and other activities. This state persists across repository shutdown, until you execute `SystemRepository resumeCommits`.

Step 2. On the slave system, when a transaction record of a checkpoint with a failover timestamp is received, the slave system will stop continuous restore. This ensures the slave has all records from the master, and no further work may be done on the master that could be lost.

At this point, you may perform the commit restore on the slave system and begin using that as your master system, and shut down the previous-master.

However, if the standby Stone was started up later than the transaction record containing a failover timestamp, it is not considered to be a failover scenario and the standby does not stop continuous restore. If you intend to do a planned failover, ensure that the slave stone is running before suspending commits for failover on the master system.

Connecting using SSL Mode

Generally, both master and slave nodes of a hot standby would be within your secure network, and benefit from the ease and performance of regular socket connections.

However, you may also configure the logsender-logreceiver connection to use SSL, if the network between your master and slave system is not secure.

When you run the hotstandby in SSL mode, you will need SSL credentials for both the logsender and logreceiver in order for them to connect.

SSL standard TLS v1.2 is used.

The SSL-specific arguments are optional, and include:

- C *fileName* certificate in PEM format that will be sent to the peer upon request.
- J *fileName* certificate authority (CA) file in PEM format to use for peer certificate verification.
- K *fileName* private key in PEM format for the certificate (-C option).
- Q *string* private key passphrase. Required if the -K option is used and the private key is encrypted.
- S enable SSL mode. Must be specified to use any other SSL options and must also be specified when starting the peer process.
- V Disable verification of the peer's certificate.

For example, to setup a logsender/logreceiver with certificate verification fully enabled, using the example certificates provided with the GemStone/S 64 Bit distribution:

```
startlogsender -A santiam.gemtalksystems.com -P 57222 -s masterStone
-T $GEMSTONE/data
-S -J $GEMSTONE/examples/openssl/certs/cacert.pem
-C $GEMSTONE/examples/openssl/certs/server_1_servercert.pem
-K $GEMSTONE/examples/openssl/private/server_1_serverkey.pem
-Q ax3vd1PI6tkQEaW9EliXW2Lbzm9qgOA1bcsDudngRAkdpA8ffwcVnA==

startlogreceiver -A santiam.gemtalksystems.com -P 57222 -s slaveStone
-T $GEMSTONE/masterTranlogs
-S -J $GEMSTONE/examples/openssl/certs/cacert.pem
-C $GEMSTONE/examples/openssl/certs/client_1_clientcert.pem
-K $GEMSTONE/examples/openssl/private/client_1_clientkey.pem
-Q rFjqh4Q71Bh+6mAH8U1tFJFvZUZkUIyLY+7A7mU0bT450c9voEvWA==
```

Self signed certificates

By default, self-signed certificates will be rejected, because the certificate cannot be verified by a known certificate authority. To use self-signed certificates, the signer must be added to the CA list in the CA file (-J flag), or certificate verification must be disabled with the -V flag. Using -V effectively tells OpenSSL to ignore certificate errors. In this mode, communications between the logsender and receiver are encrypted, but the identities of the logsender and/or logreceiver have not been verified.

12.4 Tuning a Warm or Hot Standby

The key consideration in tuning your standby system is keeping the standby as up-to-date as possible, so it will be ready if needed. This differs from the tuning priorities on your production system, which must balance user commit activity against background maintenance operations.

As your standby replays transactions, it duplicates the work that was performed on the primary system. Large operations that take considerable time on the production system will also require time on the standby.

Tuning Reclaim

When transactions from the production system are replayed, and create a large amount of reclaim work to be done, the time that this reclaim takes can be a significant bottleneck.

The parameters that are used to tune reclaim in your primary system may be set to values that ensure that reclaim does not delay user commits, which does not provide optional reclaim performance for the standby.

To tune reclaim on your standby, set the ReclaimGem configuration parameters to standby-specific values using the runtime interface.

For example, if you have set the parameter `#sleepTimeBetweenReclaimMs` to a nonzero value in your production system, to ensure that ReclaimGem activity does not block user sessions, you can reset this to zero in the standby system:

```
System setReclaimConfig: #sleepTimeBetweenReclaimMs toValue: 0
```

This will need to be executed again if you stop and restart your standby system, since it is not a persistent change; you cannot commit persistent changes to a repository in restore mode.

For a full set of Reclaim Gem configuration parameters, see “Tuning Reclaim” on page 273.

Managing Memory

Executing your application code will naturally require access to previously committed objects in the repository. These objects are faulted into the Gem session's memory to be examined or updated. When new objects are created, they must reside in memory while they are being modified.

GemStone automatically garbage-collects temporary objects that are no longer referenced, and clears out the space used by persistent, committed objects, when memory is needed. However, the memory available for any session is finite. If you need to create large temporaries or modify many objects within a transaction, you may need to tune your application to increase the available memory, or to use memory more efficiently.

This chapter discusses the following topics:

Memory Organization (page 249)

How a GemStone session's memory is organized.

Configuring Temporary Memory Usage (page 250)

How to configure temporary object memory, and debug out-of-memory problems.

13.1 Memory Organization

Each Gem session has a temporary object memory that is private to the Gem process and its corresponding session. This local object memory is divided into the following regions:

- ▶ `new` – Young temporary objects; includes two subspaces named `eden` and `survivor`
- ▶ `old` – Older temporary objects
- ▶ `pom` – Unmodified faulted-in objects, divided into ten subspaces
- ▶ `perm` – Faulted-in or created Classes and Metaclasses
- ▶ `code` – Instances of `GsNMethod` being executed, or recently executed
- ▶ `mE` – `oopMap` entries that map `objId` to in-memory objects for committed objects

Temporary objects are created in the new area of local object memory. When the new area fills up, a scavenge occurs which throws away unreferenced objects in new. After an object has survived a number of scavenges, it is copied to the old area. After the old area has grown by some amount or is almost full, a mark/sweep takes place, finding all live objects and then compacting the new, old, perm, and code areas as needed to remove dead objects.

Committed objects referenced by the session are copied from the shared page cache into the pom, perm, or code areas at the point they are first referenced by interpreter execution or a GCI call. (This is called a "copy-on-read" design.)

If a committed object in the pom area has been modified, it is copied to the old area if a scavenge occurs before the change is committed. Objects that are sent to the GCI client, such as GBS, are also moved to the old area, whether or not they are modified.

When the pom area becomes full, the contents of its oldest subspace (that is, the oldest 10%) are discarded, and that subspace is reused to continue faulting-in committed objects. Before the oldest subspace is recycled, any objects in the subspace that have been modified, or that are currently referenced from the interpreter stack, are copied to the old area.

At transaction commit, any committed objects that have been modified, and any new objects transitively reachable from those modified objects, are copied to new data pages in the shared cache. A transaction conflict check is then performed. If the commit succeeds, the in-memory state of all new objects copied to the shared cache is changed to "committed". The newly committed objects are now eligible to be removed from temporary memory by a mark/sweep or scavenge if they are no longer directly referenced from temporary objects.

13.2 Configuring Temporary Memory Usage

You may encounter an OutOfMemory error if you create too large a graph of live temporary objects at any time, or if you try to modify too many committed objects in a single transaction. OutOfMemory is a fatal error that terminates the session.

Very large numbers of Classes can also fill up temporary object memory. Any class that is referenced by message send or iteration is loaded into the perm area, and its method dictionaries, classHistory, and so on are loaded into the old area.

Persistent objects that are in the export set for a GCI client, such as GemBuilder for Smalltalk, are also moved to the old area. This includes objects that are replicated but not modified.

If you find that your application is running out of temporary memory, you can use several GemStone environment variables to help you identify which parts of your application are triggering garbage collection. Once you've done that, you can set GemStone configuration options to provide the needed memory.

Configuration Options

The values for these options are set when the gem or topaz -l process is initialized. You cannot change these values without restarting the VM. For more about these options, see the descriptions that begin on page 304.

GEM_TEMPOBJ_CACHE_SIZE

The maximum size (in KB) of temporary object memory. (This limit also applies to

linked Topaz sessions and linked GemBuilder applications.) When you only change this setting, and the other GEM_TEMPOBJ... configuration options use default values, then all of the various spaces remain in proportion to each other.

GEM_TEMPOBJ_MESPACE_SIZE

The maximum size (in KB) of the mE (Map Entries) space within temporary object memory. Unless you are trying to minimize the memory footprint on HP-UX or AIX, you should always leave GEM_TEMPOBJ_MESPACE_SIZE at its default value so that the system can calculate the appropriate value. Otherwise, you are at risk of premature OutOfMemory errors.

GEM_TEMPOBJ_OOPMAP_SIZE

The size of the hash table (that is, the number of 8-byte entries) in the objId-to-object map within temporary object memory. This option should normally be left at its default setting.

GEM_TEMPOBJ_POMGEN_SIZE

The maximum size (in KB) of the POM generation area in temporary object memory. The POM generation area holds unmodified copies of committed objects that have been faulted into a Gem, and is divided into ten subspaces. This option should normally be left at its default setting so that the system can calculate the value.

Methods for Computing Temporary Object Space

To find out how much space is left in the old area of temporary memory, the following methods in class System (category Performance Monitoring) are provided:

System _tempObjSpaceUsed

Returns the approximate number of bytes of temporary object memory being used to store objects.

System _tempObjSpaceMax

Returns the approximate maximum number of bytes of temporary object memory that are usable for storing objects.

System _tempObjSpacePercentUsed

Returns the approximate percentage of temporary object memory that is in use to store temporary objects. This is equivalent to the expression:

```
(System _tempObjSpaceUsed * 100) //
  System _tempObjSpaceMax.
```

Note that it is possible for the result to be slightly greater than 100%. Such a result indicates that temporary memory is almost completely full.

Sample Configurations

This section presents several sample configurations:

- ▶ Small configuration
- ▶ Larger old area, smaller pom
- ▶ Smaller old area, larger pom

These examples assume that you have already set the GS_DEBUG_VMGC... environment variables (see page 250) to produce the resulting printouts. The examples shown are for Solaris and may vary on other platforms.

Smaller Configuration

A value of 10000 for GEM_TEMPOBJ_CACHE_SIZE (that is, 10 MB) produces a limit of about 7 MB of temporary plus modified-committed objects (old), space for a working set of about 8 MB of unmodified committed objects (pom), and a maximum memory footprint on the order of 25 MB.

The following example shows the printout for this configuration:

```
vmGc spaceSizes: eden init 1864K max 1864K, survivor init 320K
max 320K,
vmGc old max 7496K, code max 2000K, perm max 1000K, pom 10 *
840K=8400K,
vmGc remSet 216K, meSpace max 9592K oopMapSize 65536
```

(The internal structures remSet, meSpace, and oopMapSize are not of interest here.)

Larger old, Smaller pom

The following settings configure the application for a 5 MB working set of unmodified committed objects (smaller than the default), and a maximum of 25 MB of temporary plus modified objects (larger than the default).

```
GEM_TEMPOBJ_CACHE_SIZE = 25000;
GEM_TEMPOBJ_POMGEN_SIZE = 5000;
```

The following example shows the printout for this configuration:

```
vmGc spaceSizes: eden init 2000K max 4688K, survivor init 400K
max 784K
vmGc old max 18744K, code max 5000K, perm max 2504K, pom 10 *
504K=5040K
vmGc remSet 360K, meSpace max 16824K oopMapSize 65536
```

Smaller old Area, Larger pom

The following settings configure an application with a large working set of committed objects and small temporary object space.

```
GEM_TEMPOBJ_CACHE_SIZE = 7000;
GEM_TEMPOBJ_POMGEN_SIZE = 100000;
```

The following example shows the printout for this configuration:

```
vmGc spaceSizes: eden init 1304K max 1304K , survivor init 224K
max 224K
vmGc old max 5248K, code max 1400K, perm max 704K, pom 10 *
10000K=100000K
vmGc remSet 1008K, meSpace max 48880K oopMapSize 524288
```

Debugging out-of-memory errors

When any of the following environment variables are set to a positive non-zero value, they have the effect described here for each Gem or linkable Topaz (topaz -l) process that you subsequently start. For all of these environment variables, the printout goes to the "output push" file of a linkable Topaz (topaz -l) session, for use in testing your application. If that file is not defined, the printouts go to standard output of the session's Gem or topaz -l process.

To set any of these environment variables, you must first uncomment them in the \$GEMSTONE/sys/gemnetdebug file. Be aware that the contents of

gemnetdebug are subject to change at any time. For the most current information about these and other variables, examine the *gemnetdebug* file.

GS_DEBUG_COMPILE_TRACE

Trace method compiles. The following are valid values:

- 0 - no tracing
- 1 - one line (class,selector) of each method compiled
- 2 - in addition to above, bytecode disassembly
- 3 - in addition to above, native code assembly listing

GS_DEBUG_VMGC_MKSW_MEMORY_USED_SOFT_BREAK

At the end of each mark/sweep, if the percent of memory used is greater than the threshold specified by this variable, a SoftBreak (error 6003) is generated, and the threshold is raised by 5 percent. We suggest a setting of 75%.

GS_DEBUG_VMGC_MKSW_PRINT_STACK

The mark/sweep count at which to begin printing the Smalltalk stack at each mark/sweep.

GS_DEBUG_VMGC_MKSW_PRINT_C_STACK

The mark/sweep count at which to begin printing the C stack at each mark/sweep. This variable is very expensive, consuming two seconds plus the cost of fork() for each printout.

GS_DEBUG_VMGC_PRINT_MKSW

The mark/sweep count at which to begin printing mark/sweeps.

GS_DEBUG_VMGC_PRINT_MKSW_MEMORY

The mark/sweep count at which to begin printing detailed memory usage (20 lines) for each mark/sweep.

GS_DEBUG_VMGC_PRINT_MKSW_MEMORY_USED

Specifies when Smalltalk stack printing starts as the application approaches OutOfMemory conditions. At the end of each mark/sweep, if the percent of memory used is greater than the threshold specified by this variable, the mark/sweep is printed, the Smalltalk stack is printed, and the threshold is raised by 5 percent. In a situation producing an OutOfMemory error, you should get several Smalltalk stacks printed in the Gem log file before the session dies.

GS_DEBUG_VMGC_PRINT_SCAV

The scavenge count at which to begin printing scavenges. Once this takes effect, all mark/sweeps will also be printed. Be aware that printing scavenges can produce large quantities of output.

GS_DEBUG_VM_PRINT_TRANS

Print transaction boundaries (begin/commit/abort) in the log file.

GS_DEBUG_VMGC_SCAV_PRINT_STACK

The scavenge count at which to begin printing the Smalltalk stack at each scavenge. Be aware that this print activity can produce large quantities of output.

GS_DEBUG_VMGC_SCAV_PRINT_C_STACK

The scavenge count at which to begin printing the C stack at each scavenge. This variable is very expensive, consuming 2 seconds plus the cost of fork() for each printout.

GS_DEBUG_VMGC_VERBOSE_OUTOFMEM

Automatically call the primitive for

`System class>>_vmPrintInstanceCounts:0` when an OutOfMemory error occurs, and also print the Smalltalk stack. (For details about this method, see the comments in the image.) This applies to each Gem or linkable Topaz (topaz -l) process that you subsequently start.

GS_DEBUG_VMGC_VERIFY_MKSW

The mark/sweep count at which to begin verifying object memory before and after each mark/sweep.

GS_DEBUG_VMGC_VERIFY_SCAV

The scavenge count at which to begin verifying object memory before and after each scavenge. Once this takes effect, `GS_DEBUG_VMGC_VERIFY_MKSW` will also be in effect. Be aware that this activity uses significant amounts of CPU time.

Recording Out of Memory Information to CSV file

The Out of memory and almost out of memory information that is written to a Gem log can also be directed to write to a disk log file in (CSV) comma separated format, which can be loaded into spreadsheet applications for analysis.

To configure your system to write such a CSV file, set the configuration option `GEM_TEMPOBJ_OOMSTATS_CSV` to `TRUE`.

If a Gem or linked topaz process encounters an almost out of memory or out of memory condition, a file will be created or appended to with the name `gemnetobjectpid.csv` or `topazpid.csv`.

The config parameter `GEM_TEMPOBJ_OOMSTATS_CSV` and the environment variable `GS_DEBUG_VMGC_VERBOSE_OUTOFMEM` operate independently.

Signal on low memory condition

When a session runs low on temporary object memory, there are actions it can take to avoid running out of memory altogether. By enabling handling for the signal `AlmostOutOfMemory`, an application can take appropriate action before memory is entirely full. This signal is asynchronous, so may be received at any time memory use is greater than the threshold the end of an in-memory markSweep. However, if the session is executing a user action, or is in index maintenance, the error is deferred and generated when execution returns.

When performing index operations, such as creating indexes for large collections, the `IndexManager` can be configured to use this facility to automatically commit when memory is low. Committing objects allows them to be removed from memory, since they can be re-loaded as needed from the persistent object. See the *Programming Guide* for details on `IndexManager autoCommit`, and for more information on handling `AlmostOutOfMemory`.

Managing Growth

In the course of everyday operations, your GemStone/S 64 Bit repository will grow. Some of this growth will be the result of new data in your repository, but some will represent unreferenced or outdated objects. These objects, no longer needed, must be removed to prevent the repository from growing arbitrarily large. The process of removing unwanted objects to reclaim their storage is referred to as *garbage collection*.

This chapter describes GemStone's garbage collection mechanisms and explains how and when to use them.

This chapter discusses the following topics:

Basic Concepts (page 255)

The main concepts underlying garbage collection.

Garbage Collection Operations

This include **MarkForCollection** (page 264), **Epoch Garbage Collection** (page 266), and **Reclaim** (page 272).

Running Admin and Reclaim Gems (page 275)

How to configure, start, and stop the Admin Gems and the Reclaim Gem.

Further Tuning Garbage Collection (page 278)

Tuning multi-threaded scan operations, and other special issues affecting Garbage Collection.

14.1 Basic Concepts

Smalltalk execution can produce a number of objects needed only for the moment. In addition, normal business operations can cause previously committed objects to become obsolete. To make the best use of system resources, it is desirable to reclaim the resources these objects use as soon as possible.

Different Types of Garbage

Garbage collection mechanisms vary according to *where* garbage collection occurs – temporary (scratch) memory or permanent object space – and *how* it occurs – automatically, or in response to an administrator’s action.

Each Gem session has its own private memory intended for scratch space, known as *local object memory*. The Gem session uses local object memory for a variety of temporary objects, which can be garbage-collected individually. This type of garbage collection is handled automatically by the session and is (for the most part) not configurable, although memory can be configured for specific gem requirements. These issues are covered in Chapter 13, “Managing Memory”, starting on page 249.

Permanent objects are organized in units of 16 KB called *pages*. Pages exist in the Gem’s private page cache, the Stone repository monitor’s private page cache, the shared page cache, and on disk in the extents. When first created, each page is associated with a specific transaction; after its transaction has completed, GemStone does not write to that page again until all its storage can be reclaimed.

Objects on pages are not garbage-collected individually. Instead, the presence of a shadow object or dead object triggers reclaim of the page on which the object resides. Live objects on this page are copied to another page.

The Process of Garbage Collection

Removing unwanted objects is a two-phase process:

1. Identify – *mark* – superfluous objects.
2. *Reclaim* the resources they consume.

Together, *marking* and *reclaiming* unwanted objects is *collecting garbage*.

Complications ensue because each Gem in a transaction is guaranteed a consistent view of the repository: all visible objects are guaranteed to remain in the same state as when the transaction began. If another Gem commits a change to a mutually visible object, both states of the object must somehow coexist until the older transaction commits or aborts, refreshing its view. Therefore, resources can be reclaimed only after all transactions concurrent with marking have committed or aborted.

Older views of committed, modified objects are called *shadow objects*.

Garbage collection reclaims three kinds of resources:

- ▶ The storage occupied by dead objects
- ▶ The storage occupied by shadow objects
- ▶ Object identifiers (OOPs) for dead objects

Live objects

GemStone considers an object *live* if it can be reached by traversing a path from AllUsers, the root object of the GemStone repository. By definition, AllUsers contains a reference to each user’s UserProfile. Each UserProfile contains a reference to the symbol list for a given user, and those symbol dictionaries in these lists in turn point to classes and instances created by that user’s applications. Thus, AllUsers is the root node of a tree whose branches and leaves encompass all the objects that the repository requires at a given time to function as expected.

Transitive closure

Traversing such a path from a root object to all its branches and leaves is called *transitive closure*.

Dead objects

An object is *dead* if it cannot be reached from the AllUsers root object. Other dead objects may refer to it, but no live object does. Without living references, the object is visible only to the system, and is a candidate for reclaim of both its storage and its OOP.

Shadow objects

A *shadow object* is a committed object with an outdated value. A committed object becomes shadowed when it is modified during a transaction. Unlike a dead object, a shadow object is still referenced in the repository because the old and new values share a single object identifier. The shadow object must be maintained as long as it is visible to other transactions on the system; then the system can reclaim only its storage, not its OOP (which is still in use identifying the committed object with its current value).

Commit records

Views of the repository are based on *commit records*, structures written when a transaction is committed. Commit records detail every object modified (*the write set*), as well as the new values of modified objects. The Stone maintains these commit records; when a Gem begins a transaction or refreshes its view of the repository, its view is based on the most recent commit record available.

Each session's view is based on exactly one commit record at a time, but any number of sessions' views can be based on the same commit record.

NOTE

The repository must retain each commit record and the shadow objects to which it refers as long as that commit record defines the transaction view of any session.

Commit record backlog

The list of commit records that the Stone maintains in order to support multiple repository views is the *commit record backlog*.

Shadow or Dead?

The following example illustrates the difference between dead and shadow objects. In Figure 14.1, a user creates a SymbolAssociation in the SymbolDictionary Published. The SymbolAssociation is an object (oop 27111425) that refers to two other objects, its instance variables key (#City, oop 20945153), and value ('Beaverton', oop 27110657).

The Topaz command "display oops" causes Topaz to display within brackets ([]) the identifier, size, and class of each object. This display is helpful in examining the initial SymbolAssociation and the changes that occur.

Figure 14.1 An Association Is Created and Committed

```

topaz 1> display oops
topaz 1> printit
Published at: #City put: 'Beaverton'.
Published associationAt: #City
%
[27111425 sz:2 cls: 111617 SymbolAssociation] a SymbolAssociation
  key          [20945153 sz:4 cls: 110849 Symbol] City
  value        [27110657 sz:9 cls: 74753 String]

```

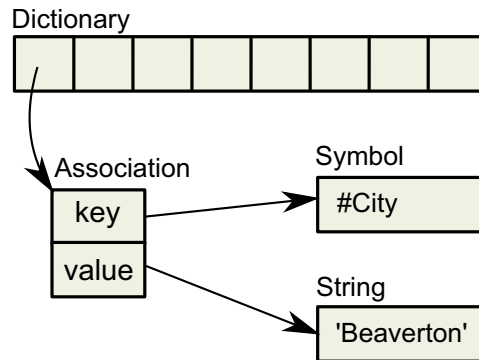


Figure 14.2 shows a second Topaz session that logs in at this point. Notice that the Topaz prompt identifies the session by displaying a digit. Because Session 1 committed the SymbolAssociation to the repository, Session 2 can see the SymbolAssociation.

Figure 14.2 A Second Session Can See the Association

```

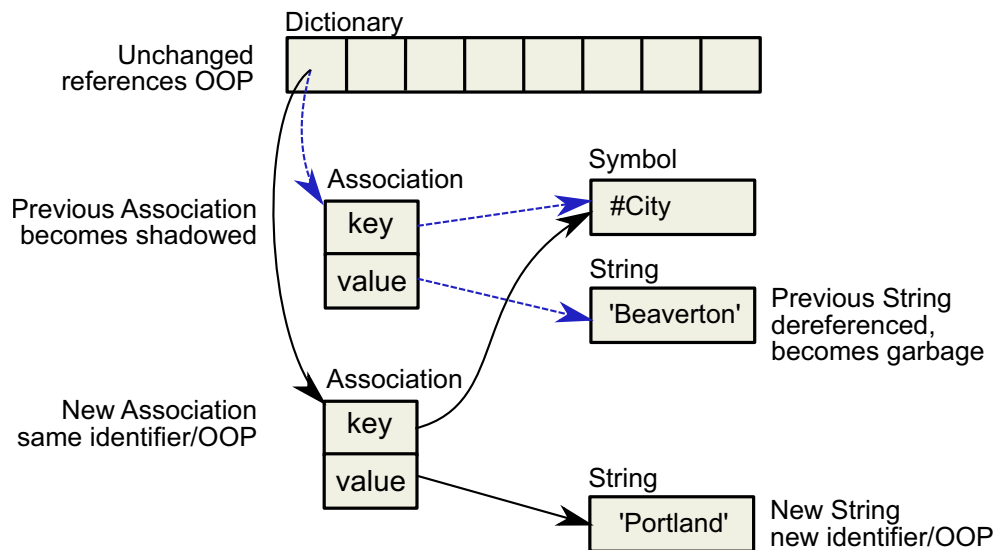
topaz 2> display oops
topaz 2> printit
Published associationAt: #City.
%
[27111425 sz:2 cls: 111617 SymbolAssociation] a SymbolAssociation
  key          [20945153 sz:4 cls: 110849 Symbol] City
  value        [27110657 sz:9 cls: 74753 String] Beaverton

```

Now Session 1 changes the *value* instance variable, creating a new SymbolAssociation (Figure 14.3). Notice in the oops display that the new SymbolAssociation object has the same identifier (27111425) as the previous Association.

Figure 14.3 The Value Is Replaced, Changing the Association

```
topaz 1> printit
City := 'Portland'.
Published associationAt: #City.
%
[27111425 sz:2 cls: 111617 SymbolAssociation] a SymbolAssociation
key [20945153 sz:4 cls: 110849 Symbol] City
value [27109121 sz:8 cls: 74753 String] Portland
topaz 1> commit
Successful commit
```



- ▶ The SymbolAssociation is now *shadowed*. Because the shadow SymbolAssociation was part of the committed repository and is still visible to other transactions (such as that of Session 2), it cannot be overwritten. Instead, the new SymbolAssociation is written to another page, one allocated for the current transaction.
- ▶ The previous value (oop 27110657) is no longer referenced in the repository. For now, this object is considered *possibly dead*; we cannot be sure it is dead because, although the object has been dereferenced by a committed transaction, other, concurrent transactions might have created a reference to it.

Even though Session 1 committed the change, Session 2 continues to see the original SymbolAssociation and its value (Figure 14.4). Session 2 (and any other concurrent sessions) will not see the new SymbolAssociation and value until it either commits or aborts the transaction that was ongoing when Session 1 committed the change.

Figure 14.4 Session 2 Sees Change After Renewing Transaction View of Repository

```
topaz 2> printit
Published associationAt: #City.
%
[27111425 sz:2 cls: 111617 SymbolAssociation] a SymbolAssociation
key [20945153 sz:4 cls: 110849 Symbol] City
value [27110657 sz:9 cls: 74753 String] Beaverton
```

```
topaz 2> abort
topaz 2> printit
Published associationAt: #City.
%
[27111425 sz:2 cls: 111617 SymbolAssociation] a SymbolAssociation
key [20945153 sz:4 cls: 110849 Symbol] City
value [27109121 sz:8 cls: 74753 String] Portland
```

Only when all sessions with concurrent transactions have committed or aborted can the shadow object be garbage collected.

What Happens to Garbage?

This section describes the steps involved in garbage collection. Specific garbage collection mechanisms will follow these steps, although the details will vary when using different garbage collection mechanisms.

The basic garbage collection process encompasses nine steps:

1. Find all the live objects in the system by traversing references, starting at the system root AllUsers. This step is called *mark/sweep*.
2. The Gem that performed mark/sweep now has a list of all live objects. It also knows the universe of all possible objects: objects whose OOPs range from zero to the highest OOP in the system. It can now compute the *set of possible dead objects* as follows:

- a. Subtract the live objects from the universe of possible objects.
- b. Subtract all the unassigned (free) OOPs in that range.

This step is called the *object table sweep* because the Gem uses the object table to determine the universe of possible objects and the unassigned OOPs.

3. The Gem performing this work now has a list of *possibly dead* objects. We can't be sure they're dead because, during the time that the mark/sweep and object table sweep were occurring, other concurrent transactions might have created references to some of them.

The Gem sends the Stone the *possible dead set* and returns.

4. Now, in a step called *voting*, each Gem logged into the system must search its private memory to see if it has created any references to objects in the possible dead set. When it next commits or aborts, it votes on every object in the possible dead set. Objects referenced by a Gem are removed from the possible dead set.

NOTE

Gems do not vote until they complete their current transaction. If a Gem is sleeping or otherwise engaged in a long transaction, the vote cannot be finalized

and garbage collection pauses at this point. Commit records accumulate, garbage accumulates, and a variety of problems can ensue.

5. Because all the previous steps take time, it's possible that some Gems were on the system when the mark/sweep began, created a reference to an object now in the possible dead set, and then logged out. They cannot vote on the possible dead set, but objects they've modified are in the write sets of their commit records. The *Admin Gem*, a process dedicated to administrative garbage collection tasks, scans all these write sets (the *write set union*), and votes on their behalf. This is called the *write set union sweep*.
6. After all voting is complete, the resulting set now holds nothing but unreferenced objects. The Stone now promotes the objects from possibly dead to dead.
7. the *Reclaim Gem* reclaims pages: it copies live objects on the page onto a new page, thereby compacting live objects in page space. The page now contains only recycleable objects and perhaps free space.
8. The Reclaim Gem commits. The reclaimed OOPs are returned to their free pool.
9. The Reclaim Gem's commit record is disposed of. The reclaimed pages are returned to their free pool.

Admin and Reclaim Gems

It is useful to understand the distinction between the Admin Gem and the Reclaim Gem:

- ▶ The Admin Gem finalizes the vote on possibly dead objects (Step 5 on page 261), and performs the write set union sweep. The Admin Gem also performs Epoch Garbage Collection (page 266), if enabled.
- ▶ The Reclaim Gem is dedicated to the task of reclaiming shadowed pages and dead objects repository-wide, along with their OOPs.

The Reclaim Gem includes a master session and multiple reclaim sessions, each being a thread within the Reclaim Gem process. This allows reclaim to occur in parallel.

By default, the Admin Gem and the Reclaim Gem with one reclaim session are configured to run, and are started automatically when the Stone is started. By default, epoch is disabled.

- ▶ We recommend that you leave the Admin Gem running at all times, although it is required only following a `markForCollection` or `markGcCandidatesFromFile:`, or after a epoch garbage collection operation. (Subsequent sections of this chapter describe these operations in detail.) If the Admin Gem is not running following one of these operations, the garbage collection process cannot complete, and garbage can build up in your system.
- ▶ We recommend that you have the Reclaim Gem running at all times, to reclaim shadow objects.

Admin and Reclaim Gem configuration parameters

Both the Admin and Reclaim Gems are run from the GcUser account, a special account that logs in to the repository to perform garbage collection tasks. This account is used to set configuration values for the GcGems.

The configuration parameters that apply to either the Admin or Reclaim Gems can either be set persistently, or at runtime.

To modify configuration parameters persistently, log in as GcUser (GcUser's initial password is 'swordfish', as is DataCurator's and SystemUser's) and send the message `at:aKey put:aValue` to UserGlobals.

For example, to set `#reclaimMinPages` to 100:

```
topaz> set user GcUser password thePassword
login
...
topaz 1> printit
UserGlobals at: #reclaimMinPages put: 100.
System commitTransaction
%
```

To set parameters at runtime, you do not need to log in as GcUser or perform a commit. A user with GarbageCollection privilege, such as DataCurator, can execute System class methods `setReclaimConfig:toValue:` or `setAdminConfig:toValue:`.

For example:

```
topaz> set user DataCurator password thePassword
login
...
topaz 1> printit
System setReclaimConfig: #reclaimMinPages toValue: 100.
%
```

Specific configuration parameters and how to apply them are discussed in detail later in this chapter.

GemStone's Garbage Collection Mechanisms

GemStone provides the following mechanisms that together mark and reclaim garbage, thereby helping you to control repository growth.

Marking

Repository-wide marking – To prevent the repository from growing large enough to cause problems on a regular basis, you can run `Repository >> markForCollection`. This method combines a full sweep of all objects in the repository and the marking of each possible dead object in a single operation.

Epoch garbage collection – If enabled, the Admin Gem periodically examines all transactions written since a specific, recent time (the beginning of this *epoch*) for objects that were created and then dereferenced during that period. However, epoch garbage collection cannot reclaim objects that are created in one epoch but dereferenced in another. In spite of its name, epoch garbage collection only marks; it does not reclaim. You can configure various aspects to maximize its usefulness. Epoch garbage collection is disabled by default. For details about epoch garbage collection, see "Epoch Garbage Collection" on page 266.

Reclaiming

Reclaim – Once you've run `markForCollection` or epoch garbage collection, the Reclaim Gem will reclaim pages that contain either dead or shadow objects. When there are a high number of objects needing to be reclaimed, you may increase the number of sessions under the Reclaim Gem. For details about reclaiming pages, see "Reclaim" on page 272.

GcLock

Many garbage collection process, such as mark/sweep operations, should not be run concurrently. To prevent this, there is a shared internal lock called the GcLock. Garbage collection processes that cannot run concurrently, such as `markForCollection`, get the GcLock, which prevents another one from starting up. The GcLock is also held by the Admin Gem at certain periods.

In addition to garbage collection, some repository-wide operations such as `GsObjectInventory` (see page 136) also hold the GcLock while they are running.

If another task that requires the GcLock is in progress at the time you try to do `markForCollection` or `findDisconnectedObjects...`, they will not execute, but report an error similar to that shown below.

```
-- Request for MFC gclock by session 10 denied, reason: vote state
is voting, sessionId not voted 2
ERROR 2501 , a Error occurred (error 2501), Request for gcLock
timed out.
'Request for MFC gclock by session 10 denied, reason: vote state
is voting, sessionId not voted 2' (Error)
```

The cause of the conflict may be:

- ▶ Another operation that requires the GcLock is in progress in another session; this includes epoch and MFC, and also operations such as `GsObjectInventory`.
- ▶ A previous epoch or `markForCollection` completed the mark phase, but voting on possibly dead objects has not completed.

For voting to complete, the Admin Gem must be running. Also, any long-running session that neither aborts nor commits will prevent the vote from completing.

Symbol Garbage Collection

Symbols in GemStone are a special case of Object, since they must always have a unique OOP across all sessions. To ensure this, symbol creation is managed by the `SymbolUser`, who creates all new Symbols. Symbols are stored in the `AllSymbols` dictionary, and are not removed, to avoid any risk of creating duplicate symbols.

However, there are cases where a large number of unimportant symbols are created, perhaps inadvertently. To reclaim this space and to manage the size of `AllSymbols`, you can configure GemStone to collect unreferenced symbols in a multi-step process that ensures that symbols in use are not collected.

By default, Symbol garbage collection is not enabled. It can be enabled using the configuration parameter, `STN_SYMBOL_GC_ENABLED`, or by the runtime equivalent, `#StnSymbolGcEnabled`. If enabled, symbol garbage collection is performed automatically in the background and requires no management.

When enabled, unused symbols are located and put in a possibleDeadSymbols collection as part of a markForCollection. These symbols are hidden, to remove references from AllSymbols but retain the OOPs until the voting, union, and finalization is done. Any lookups on the hidden symbol will return the existing hidden symbol and restore it to the AllSymbols dictionary.

Once voting and write-set union sweep are done, the symbols that are otherwise unreferenced are removed from the possibleDeadSymbols, so they will be collected by the next markForCollection.

14.2 MarkForCollection

Privileges required: GarbageCollection.

The method `Repository>>markForCollection` sweeps the entire repository and marks as live all objects that can be reached through a transitive closure on the symbol lists in AllUsers, as described on page 256. The remaining objects become the list of possible dead objects.

`markForCollection` only provides a set of possible dead objects for voting and eventual reclaiming as described under “What Happens to Garbage?” on page 260. It does not reclaim the space or OOPs itself; the Reclaim Gem does that, as described under “Reclaim” on page 272.

To mark unreferenced GemStone objects for collection, log in to GemStone and send your repository the message `markForCollection`, as in the following example:

```
topaz 1> printit
SystemRepository markForCollection
%
```

If you are performing `markForCollection` on a large production repository, consider the steps described under “Impact on Other Sessions” on page 265.

This method aborts the current transaction and runs `markForCollection` inside a transaction, but monitors the commit record backlog so it can abort as necessary to prevent the backlog from growing. When `markForCollection` completes, the session reenters a transaction, if it was in one when this method was invoked.

When `markForCollection` completes successfully, the Gem that started it displays a message such as the one below:

```
Warning: a Warning occurred (notification 2515), markForCollection
found 110917 live objects, 3496 dead objects(occupying approx
314640 bytes)
```

If another garbage collection task is in progress at the time you try to do `markForCollection`, this method will retry for a fixed period, reporting status. If the other operation does not complete within the timeout period, it reports an error indicating it could not get the GcLock. See “GcLock” on page 263 for more details.

Before issuing the error, the `markForCollection` method waits up to a minute for the other operation to complete. To have the `markForCollection` wait for a longer period, use `markForCollectionWait: waitTimeSeconds`. To wait as long as necessary for the other garbage collection to finish, pass the argument `-1`. Do so with caution, however; under certain conditions, the session could wait forever. To avoid this:

- ▶ Make sure that other sessions are committing or aborting, which allows voting on possible dead to complete.
- ▶ Make sure that the Admin Gem is running to complete processing of dead objects once the vote is completed.

Impact on Other Sessions

The `markForCollection` operation uses multi-threaded scan. For more details on this, see “Multi-Threaded Scan” on page 278.

By default, `markForCollection` limits its use of CPU resources if the CPU load on the system reaches 90%. It starts the operation with two threads and a page buffer size of 128. If the CPU limit is reached, the code automatically causes threads to sleep until the load is less than 90%. Depending upon the I/O required, the system may never reach this limit.

To enable `markForCollection` to complete as quickly as possible, you can use:

```
SystemRepository fastMarkForCollection
```

This uses higher settings (95% of CPU, and a number of threads based on the current hardware) to use as many system resources as possible. The performance of anything else running on the same system may be heavily degraded.

For maximum control, use the method

```
SystemRepository markForCollectionWithMaxThreads: threadsCt
  waitForLock: seconds
  pageBufSize: pageBufSize
  percentCpuActiveLimit: percentLimit
```

This allows you to specify the precise limits.

Starting `markForCollection` with these limits provides a specification for the trade-off you wish to make between speed to complete and the impact on other sessions. The desired trade-off may vary over time; for example, if your `markForCollection` extends over both business hours and non-business hours, you may accept greater impact during these periods of light load. The Multi-threaded scan parameters can be changed at runtime, as described under “Tuning Multi-Threaded Scan” on page 278.

After the `markForCollection` has completed, there may be additional impact on other sessions, since it is likely that dead objects that require reclaim were identified. After the remaining Garbage Collection steps have completed, the Reclaim Gem Sessions may become busy reclaiming the dead objects.

Scheduling markForCollection

To invoke `markForCollection` using the **cron** facility, create a three-line script file similar to the Topaz example on page 264 by entering everything except the prompt. Use this script as standard input to **topaz**, and redirect the standard output to another file:

```
topaz < scriptName > logName
```

Make sure that `$GEMSTONE` and any other required environment variables are defined during the **cron** job. Either create a `.topazini` file for a user who has GarbageCollection privilege, or insert those login settings at the beginning of the script. For information about using **cron**, refer to your operating system documentation.

14.3 Epoch Garbage Collection

Privileges required: `GarbageCollection`.

Epoch garbage collection operates on a finite set of recent transactions: the *epoch*. Using the write set that the Stone maintains for each transaction, the Admin Gem examines every object created during the epoch. If an object is unreferenced by the end of the epoch, it is marked as garbage and added to the list of possible dead objects.

Epoch collection is efficient because:

- ▶ It's faster and easier to perform a transitive closure on a few recent transactions than on the entire repository.
- ▶ Most objects die young, especially in applications characterized by numerous small transactions updating a few previously committed objects. An epoch of the right length can collect most garbage automatically.

Although epoch collection identifies a lot of dead objects, it cannot replace `markForCollection` because it will never detect objects created in one epoch and dereferenced in another.

By default, epoch garbage collection is disabled. You can enable it in either of two ways:

- ▶ Before you start the Stone, set the `STN_EPOCH_GC_ENABLED` configuration option to `TRUE`.
- ▶ Execute the method `System class>>enableEpochGc`. You may also manually disable epoch garbage collection using `System class>>disableEpochGc`. Using these methods updates the system configuration file.

After your installation has been operating for a while, and you've had the chance to collect operational statistics, consider this: epochs of the wrong length can be notably inefficient. The section "Determining the Epoch Length" on page 267 includes an in-depth discussion of the performance trade-offs of short or long epochs

Running Epoch Garbage Collection

When epoch garbage collection is enabled, it will run automatically according to the `GcUser` configuration parameters `#epochGcTimeLimit` and `#epochGcTransLimit`.

You can force an epoch garbage collection to begin using `System class >> forceEpochGc`. `forceEpochGc` will return `false`, and not start an epoch garbage collection, if any of the following are true:

- ▶ Checkpoints are suspended.
- ▶ Another garbage collection operation is in progress.
- ▶ Unfinalized possible dead objects exist (that is, `System voteState` returns a non-zero value).
- ▶ The system is in restore mode.
- ▶ The Admin Gem is not running.
- ▶ Epoch garbage collection is disabled (that is, `STN_EPOCH_GC_ENABLED = FALSE`).
- ▶ The system is performing a `reclaimAll`.

- ▶ A previous `forceEpochGc` operation was performed and the epoch has not yet started or completed.

Tuning Epoch

Epoch Configuration Parameters

The following configuration parameters are available to control the performance of epoch garbage collection, and are stored in `GcUser's UserGlobals`. For details on modifying values, see page 261.

<code>#epochGcTimeLimit</code>	The maximum frequency of epoch garbage collection (in seconds). Default: one hour (3600 seconds). This value should be at least 1800 (30 minutes), since the aging of objects faulted into Gem memory uses 5 minute aging for each of 10 subspaces of the POM generation.
<code>#epochGcTransLimit</code>	The minimum number of transactions required to trigger epoch garbage collection. Default: 5000.
<code>#epochGcPercentCpuActiveLimit</code>	Limit active epoch threads when system <code>percentCpuActive</code> is above this limit. Default: 90.
<code>#epochGcPageBufferSize</code>	Size in pages of buffer used for epoch GC (must be power of 2). Default: 64.
<code>#epochGcMaxThreads</code>	The <code>MaxThreads</code> used for next epochGc.

Epoch garbage collection uses the multi-threaded scan (see “Multi-Threaded Scan” on page 278) and can be tuned to complete more quickly with more system performance and resources impact, or take longer and use fewer system resources. The parameters `#epochGcPercentCpuActiveLimit`, `#epochGcPageBufferSize`, and `#epochGcMaxThreads` are used to tune epoch garbage collection’s multi-threaded scan impact.

Determining the Epoch Length

Epoch garbage collection’s ability to identify unreferenced objects depends on the relationship between three variables:

- ▶ The rate of production R of short-lived objects.
- ▶ The lifetime L of these objects.
- ▶ The epoch length E .

The only variable under your direct control is epoch length. Although you cannot specify it explicitly, the following configuration parameters jointly control the length of an epoch:

- ▶ `#epochGcTimeLimit`
- ▶ `#epochGcTransLimit`

Epoch garbage collection occurs when:

```
(the time since last epoch > epochGcTimeLimit ) AND  
(transactions since last epoch > epochGcTransLimit)
```

The following discussion assumes that the epoch is determined by the minimum time interval (#epochGcTimeLimit) because other threshold is always met.

Figure 14.5 shows the effect of the epoch on the number of items marked. If $L = E$, for example, five minutes, every object's lifetime spans epochs (top part of graph), and none are collected.

When the epoch is longer than an average object's lifetime, however, some objects live and die within the same epoch, and can be marked. The lower part of Figure 14.5 shows an example where $E = 3L$ and objects are created at a uniform rate. Objects created during the first two-thirds of the interval die before its end and are marked. Only those created during the final third survive to the next epoch.

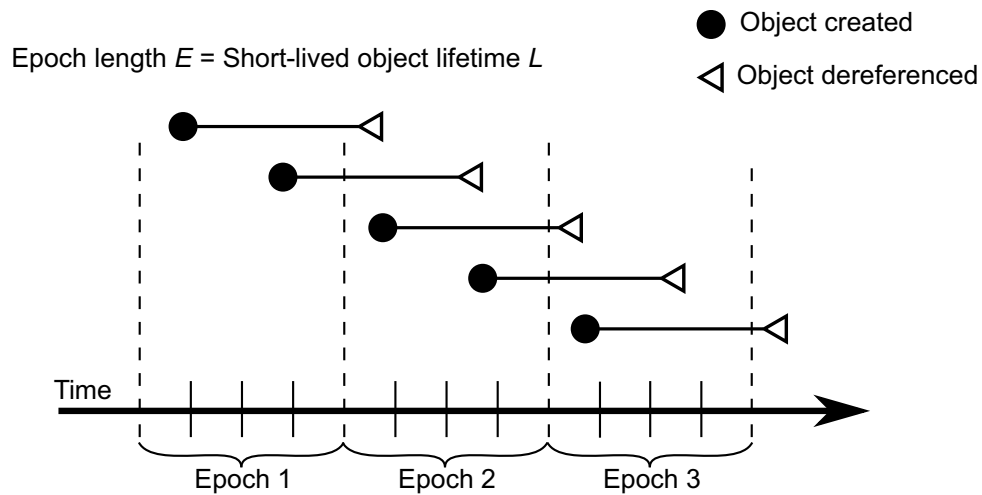
The results shown in Figure 14.5 can be expressed as:

```
Objects Missed by EpochGC =  $R \times L$   
Objects Recovered by EpochGC =  $R(E - L)$ 
```

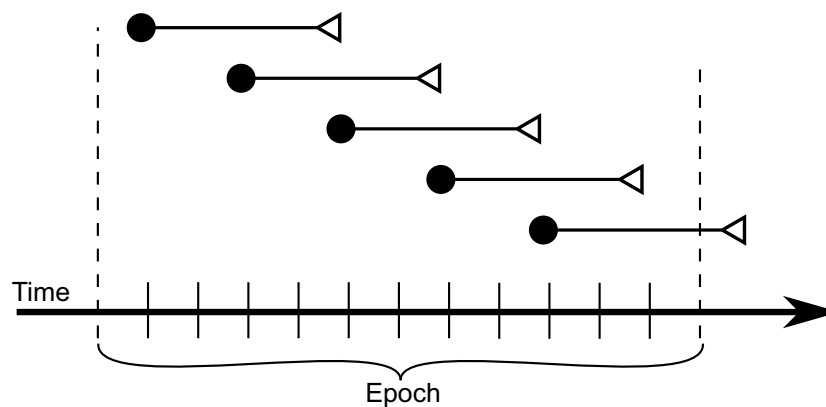
For example, assume $R = 1000$ objects per minute, $L = 5$ minutes, and $E = 15$ minutes. Then, for each epoch:

```
Objects Missed =  $1000 \times 5 = 5000$   
Objects Recovered =  $1000 (15 - 5) = 10000$ 
```

Figure 14.5 Effect of Collection Interval on Epoch Garbage Collection



Epoch length $E = 3 \times$ Short-lived object lifetime L



Therefore:

- ▶ Set `#epochGcTimeLimit` $E >$ lifetime L of short-lived objects.

Figure 14.6 graphs the effect of the epoch. When $E = L$, epoch garbage collection is in effect disabled; all objects survive into the next epoch; the number of unmarked yet dead objects in the repository grows at the creation rate. These dead objects remain unidentified until you run `markForCollection`.

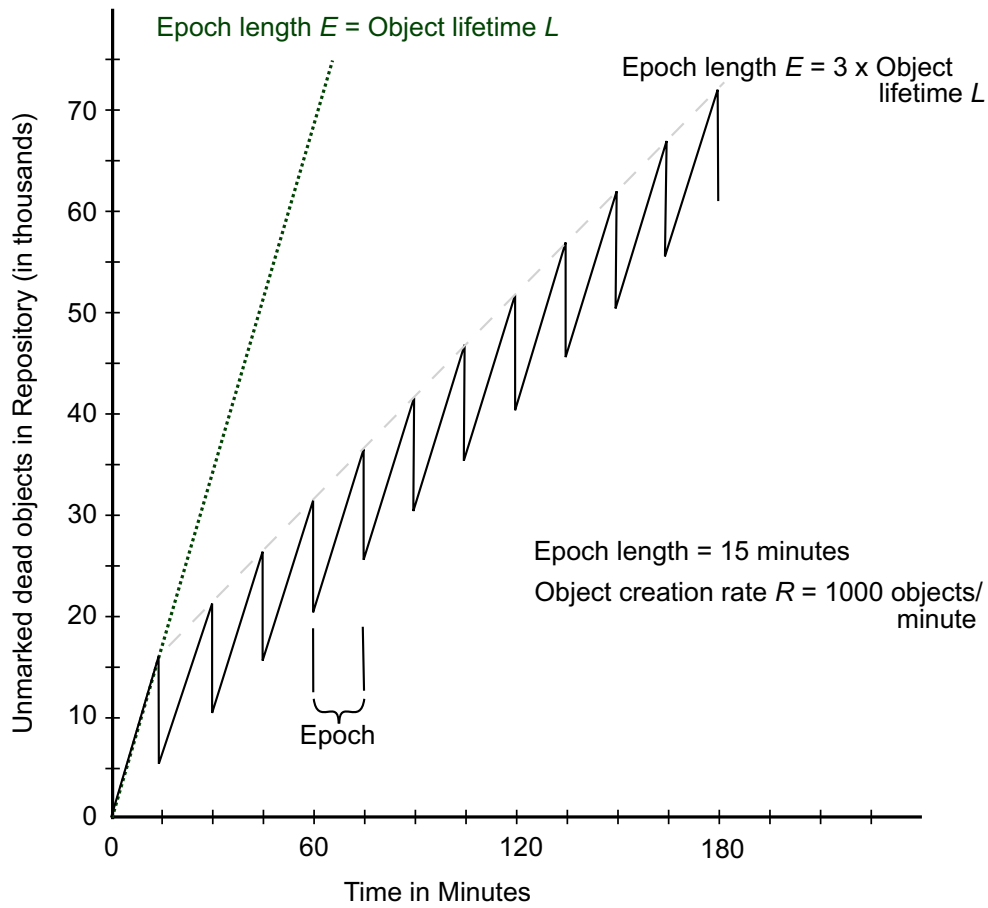
When the epoch is extended so that $E = 3L$, each epoch garbage collection marks those objects both created and dereferenced during that interval. This ratio causes the sawtooth pattern in the graph. If the creation rate is uniform, two-thirds of the dead objects are marked $((E-L)/E)$, and one-third are missed (L/E) . Consequently, the repository grows at one-third the rate of the case $E = L$.

This configuration trades short bursts of epoch garbage collection activity for:

- ▶ moderate growth in the repository, and

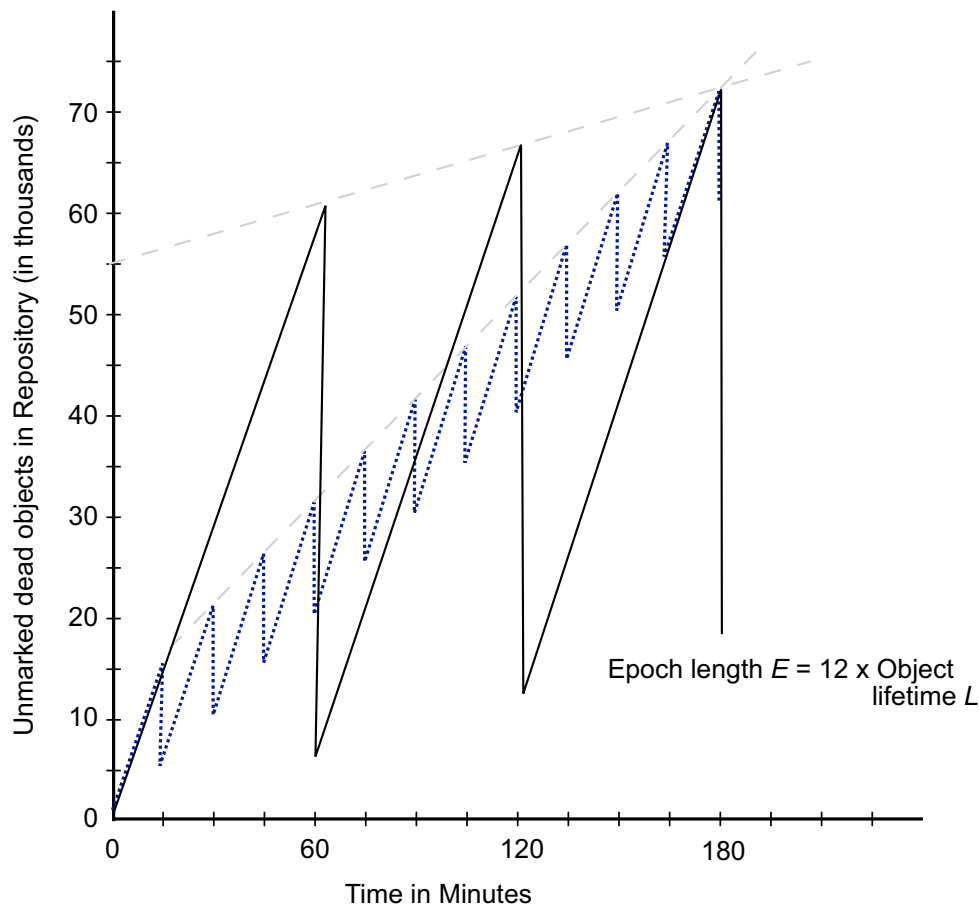
- ▶ the need to run `markForCollection` often enough to mark dead objects that survive between epochs.

Figure 14.6 Repository Growth with Short Epoch



Suppose we extend the epoch to $E = 12L$. The result is shown in Figure 14.7, superimposed on part of the previous figure.

Figure 14.7 Effect of Longer Epoch on Repository Growth



Although the longer epoch allows many more dead objects to accumulate, the growth rate of the repository is substantially less—25% of the previous case.

This configuration trades a slower growth rate for:

- ▶ a need for greater headroom on the disk, and
- ▶ longer bursts of epoch garbage collection activity.

Certain cases have needed an epoch as long as several hours, or even a day.

Cache Statistics

Several cache statistics include information about the epoch garbage collection process. These are visible by using statmonitor data viewed in VSD (the visual statistics display tool). You may also access methods in System to get the values programmatically; see “Programmatic Access to Cache Statistics” on page 138 for more information.

The following statistics may be useful in monitoring epoch:

EpochGcCount	The number of times that the epoch garbage collection process was run by the Admin Gem since the Admin Gem was started. For a system in steady state, look for uniform periods between runs or a uniform run rate.
EpochNewObjs	The number of new objects that were created during the last epoch.
EpochPossibleDeadObjs	The number of possible dead objects found by the last epoch garbage collection.
EpochScannedObjs	The number of objects scanned by the last epoch garbage collection.

14.4 Reclaim

The Reclaim Gem is responsible for reclaiming both dead and shadowed objects (see “Shadow or Dead?” on page 257 for the difference between these types of garbage).

Shadowed objects are created naturally as your application modifies existing objects, so it is a good idea to always have the Reclaim Gem running to avoid shadowed objects accumulating. Some operations, such as migration, create a very large number of shadowed objects that need to be reclaimed.

After a mark/sweep operation — markForCollection or epoch — completes, there will be a number of dead objects that need to be reclaimed.

Although it is objects that are dead or shadowed, reclaim is done in pages. Pages that contain dead or shadowed objects may also contain some live objects; these live objects are copied to fresh pages, and the resulting page may then be reclaimed.

Reclaim is performed multi-threaded. Each thread within the Reclaim Gem is similar to a session, but runs within the Reclaim Gem process.

When the Reclaim Gem is running, its sessions examine pages marked reclaimable because they contain either dead or shadow objects, and reclaim fragments of space left by transactions that did not fill an entire page. This occurs in the background, with no specific action required.

Although it is recommended to allow the background processes to perform the reclaim, you can explicitly invoke it by executing:

```
SystemRepository reclaimAll
```

Reclaimed space does not appear as free space in the repository until other sessions have committed or aborted all transactions concurrent with the reclaim transaction, and the Stone has disposed the commit record. If other users are logged in and holding up this process, you can determine which sessions are viewing the oldest commit record, thereby impeding reclaim. See the discussion under “Further Tuning Garbage Collection” on page 278.

Tuning Reclaim

Reclaim Configuration Parameters

The following configuration parameters are available to control the reclaim task, and are stored in GcUser's UserGlobals. For details on modifying values, see page 261.

<code>#deadObjsReclaimedCommit-Threshold</code>	The maximum number of dead objects to reclaim in a single transaction, including dead objects reclaimed when reclaiming shadow pages. The default is 20000.
<code>#deferReclaimCacheDirtyThreshold</code>	If the primary shared page cache (the shared cache on the stone's machine) is more than this percentage dirty, then reclaim gems will wait until the cache is less than 5% below this threshold before resuming reclaims. The default is 75%.
<code>#maxTransactionDuration</code>	The maximum length (in seconds) of a GcGem transaction. The transaction will be committed once this time is exceeded. Must be ≥ 10 ; the default is 300, maximum is <code>SmallInteger maximumValue</code> .
<code>#objectTableCleanupIdleTime</code>	The time in seconds that the system is idle (no commits occur), before the reclaim gem performs an internal <code>objectTableCleanup</code> (which involves pushing deltas to object table leaf nodes). The default value, 0, means disable the cleanup operation.
<code>#objsMovedPerCommitThreshold</code>	The approximate maximum number of live objects to move in a reclaim transaction. Must be ≥ 100 ; the default is 20000, maximum is <code>SmallInteger maximumValue</code> .
<code>#reclaimDeadEnabled</code>	A Boolean indicating whether or not to reclaim dead objects; the default is true.
<code>#reclaimMinFreeSpaceMb</code>	Minimum repository free space which must be available in order for reclaims to proceed. Reclaims will be temporarily suspended if the repository free space drops below this threshold. The default value of 0 specifies a limit computed as the current size of the repository divided by 1000, with a minimum value of 5MB. Default and minimum 0, maximum 65536.
<code>#reclaimMinPages</code>	The minimum number of pages to process in a single reclaim operation (reclaiming does not start until this threshold is reached). Must be ≥ 1 ; the default is 30 pages, maximum is <code>SmallInteger maximumValue</code> .

<code>#sleepTimeBetweenReclaimMs</code>	The minimum amount of time in milliseconds that the process will sleep between reclaims, even when work is scheduled. The default is 0 milliseconds, maximum 3600000.
<code>#sleepTimeWithCrBacklogMs</code>	Amount of time (in milliseconds) to sleep after a commit when the commit record backlog is larger than $1.25 * \text{the current setting for STN_CR_BACKLOG_THRESHOLD}$. For each 25 percent above the threshold the sleep time is increased so that the <code>reclaimGem</code> does fewer commits the higher the number of commit records is about the threshold. Must be between 0 and 300000; the default is 0.

Reclaim Commit Frequency

A Reclaim Gem session will commit reclaim changes as soon as any one of the following conditions is met:

- ▶ Number of live objects moved exceeds `#objsMovedPerCommitThreshold`.
- ▶ Duration of the transaction exceeds `#maxTransactionDuration`.
- ▶ Number of dead objects reclaimed exceeds `#deadObjsReclaimedCommitThreshold`.

Controlling the impact of reclaim

Reclaim, particularly with a larger number of sessions configured for the Reclaim Gem, can perform quickly but place a large load on your system. If you are likely to be doing reclaim during periods where users will also need to use the system, you may wish to slow down reclaim. This can be done in a number of ways:

- ▶ Reduce the number of reclaim sessions using `System class >> changeNumberOfReclaimGemSessions:` with a argument of 1 or 0.
- ▶ Set `#sleepTimeBetweenReclaimMs` to ensure that reclaim Gem sessions pause between reclaim operations.
- ▶ Set `#sleepTimeWithCrBacklogMs` so that in case your system encounters a commit record backlog, the impact of reclaim is automatically reduced.

Speeding up reclaim

You can also setup your system to run reclaim with the maximum impact during off-hours. If you have a large amount of reclaim to perform, this allows the reclaim to finish more quickly. You can increase the number of Reclaim session to the maximum using:

```
System class >> startMaxReclaimGemSessions
```

This will start the number of sessions specified by `STN_MAX_GC_RECLAIM_SESSIONS`.

Avoiding disk space issues

Reclaim requires pages from the repository in order to copy non-dead objects. There are further steps that the stone must complete, before the space on the reclaimed pages is available again. So initially, reclaim will cause the amount of free space in the repository to drop.

Depending on overhead required by your system and the largest amount of reclaim that needs to be done at any time, you may want to configure a larger `#reclaimMinFreeSpaceMb`. This will ensure that reclaim pauses before your repository becomes dangerously low in free space.

Cache Statistics

Several cache statistics provide information about reclaim. These are visible by using statmonitor data viewed in VSD (the visual statistics display tool). You may also access methods in System to get the values programmatically; see "Programmatic Access to Cache Statistics" on page 138 for more information.

The following Stone statistics may be useful in monitoring reclaim:

DeadNotReclaimedObjs	The number of objects known to be dead but not yet reclaimed.
DeadObjsReclaimedCount	The total number of dead objects reclaimed since the Stone repository monitor process was last started.
GcVoteState	Indicates the current phase of garbage collection: Gems voting, voting complete, Possible Dead Write-Set Union Sweep (PDWSUS) in progress, or PDWSUS complete.
PagesNeedReclaimSize	The amount of work waiting for the reclaim task.
PossibleDeadObjs	The number of objects marked as dereferenced but not yet declared to be dead.
ReclaimCount	The number of times the reclaim process has been run.
ReclaimedPagesCount	The number of scavenged pages.

14.5 Running Admin and Reclaim Gems

Admin Gem Privileges required: GarbageCollection

The initial configuration for the Admin and Reclaim Gems are provided in the system configuration file for the stone; by default, `$GEMSTONE/data/system.conf`. These settings determine what is started automatically when the stone starts up. During runtime, you can start and stop the Admin Gem and change the number of Reclaim sessions that are running.

Configuring Admin Gem

The Admin Gem is enabled or disabled by the setting for the `STN_ADMIN_GC_SESSION_ENABLED` configuration option. By default, this is enabled, and normally you should leave this enabled. You can stop and restart the Admin Gem at runtime as needed.

Configuring Reclaim Gem

The number of Reclaim sessions is set by the `STN_NUM_GC_RECLAIM_SESSIONS` configuration option. By default, this is one, and you should normally keep at least one Reclaim session running. Most systems will benefit from increasing the number of Reclaim sessions. In general, we recommend running one Reclaim session for between 5 and 10 extents. You may need to experiment to find the correct balance for your system. The number of Reclaim sessions can be changed at runtime as needed.

To ensure that Reclaim sessions do not impact the number of user sessions, a separate configuration setting, `STN_MAX_GC_RECLAIM_SESSIONS`, configures the maximum number of Reclaim sessions you will be running.

By default, this is set to the number of extents on your system. This parameter cannot be changed without restarting the stone. The upper limit for the number for the number of Reclaim sessions that can be run under any configuration is 255.

While the number of Reclaim sessions should normally be less than or equal to `STN_MAX_GC_RECLAIM_SESSIONS`, it is possible to start a larger number of Reclaim sessions. However, this will reduce the number of user sessions that can login to this Stone. If your system does not have excess unused user sessions, you should be careful to configure `STN_MAX_GC_RECLAIM_SESSIONS` high enough that you will never want to run a larger number of Reclaim sessions.

Starting GcGems

You can ensure all configured GcGems are running using:

```
System startAllGcGems
```

If the Admin Gem is not running, start it. If the Reclaim Gem is not running, start it with the configured number of Reclaim sessions. Return true if the Admin Gem and at least one Reclaim sessions are started.

or by executing both:

```
System startAdminGem
```

If the Admin Gem is not running, start it. Return true if the Admin Gem is running, false if the Admin Gem could not be started.

```
System startReclaimGem
```

If the Reclaim Gem is not running, start it with the configured number of Reclaim sessions. Return the number of Reclaim sessions that will be running. If the Reclaim Gem is already running, has not effect and returns the number of Reclaim sessions already running.

It may take a little time for the GcGems to complete login. The above methods do not block; they initiate the startup and return immediately. To wait a given period of time for the GcGems to start up:

```
System waitForAllGcGemsToStartForUpToSeconds: anInt
```

If the Admin Gem is not running, start it. If the Reclaim Gem is not running, start it with the configured number of Reclaim sessions. If all the GcGems have not started up within that time, return false. However, this does not necessarily mean that any GcGems have failed to start; on a slow system with a short timeout, this method may return false, even though all GcGems eventually start correctly.

To confirm that the GcGems are running,:

```
System hasMissingGcGems
```

Returns false if either the Admin Gem or the Reclaim Gem is not running.

To determine the number of Reclaim sessions that are currently running:

```
System reclaimGcSessionCount
```

Returns the total number of Reclaim sessions that are running.

Stopping GcGems

To ensure that the Admin Gem and all Reclaim sessions are stopped:

```
System stopAllGcGems
```

or you may execute both:

```
System stopAdminGem
```

```
System stopReclaimGem
```

Adjusting the number of Reclaim sessions

You can adjust the number of Reclaim sessions that are running during the course of operation of your application. When there is a large amount of reclaim and little other load on your system, running a large number of Reclaim sessions will allow the reclaim work to complete more quickly. During normal operation, reducing the number of Reclaim sessions avoids using too many system resources and impacting users.

To set the number of Reclaim sessions that are running:

```
System changeNumberOfReclaimGemSessions: targetReclaimSessionCount
```

Start the ReclaimGem, if it is not running, with *targetReclaimSessionCount* Reclaim sessions.

targetReclaimSessionCount should be a number less than or equal to the value for `STN_MAX_GC_RECLAIM_SESSIONS`. Using a larger argument does not error, but may have consequences for user logins; see the discussion on page 276.

Return the new target number of Reclaim sessions; Reclaim sessions will be started or stopped to reach this number. This method does not block, so it may take a little time before the correct number of Reclaim sessions is actually running.

Using this method only changes the currently running number of Reclaim sessions, but does not affect the configured number. After stopping the ReclaimGem, on restart the regular configured number of sessions will be started.

To change the default number of Reclaim sessions that will be started by default when the ReclaimGem starts up:

```
System configurationAt: #StnNumGcReclaimSessions
```

```
put: targetReclaimSessionCount.
```

This does not effect the number of Reclaim sessions that are currently running, if any. Changes to the runtime parameter do not persist if the Stone is restarted. For a permanent change, you should edit the configuration parameters in the configuration file used by the stone: `STN_NUM_GC_RECLAIM_SESSIONS`, and if necessary, `STN_MAX_GC_RECLAIM_SESSIONS`.

14.6 Further Tuning Garbage Collection

Multi-Threaded Scan

For large systems, it can take a considerable amount of time to scan the entire repository, as is required by a mark/sweep operation (or other operations such as `listInstances`). To allow these scans to complete faster, operations that scan the entire repository use multiple threads running in parallel. There is a trade-off between how fast the operation completes and how much of the system resources it uses. Obviously, the faster the scan completes, the less of anything else can be done on that system during that period.

The trade-off can be configured per operation and can be changed as the operation proceeds. You can choose to run it to complete as quickly as possible but use all the system resources, or with minimal impact on the rest of your application, but taking much longer to complete. You can switch between these approaches as often as you need to.

Tuning Multi-Threaded Scan

Each session has two variables that control the impact of the multi-threaded operation it is running:

<code>MtThreadsLimit</code>	The upper limit on the number of threads can be activated.
<code>MtPercentCpuActiveLimit</code>	The total CPU load level at which the scan starts to deactivate threads.

These variables are arguments to all scan operations, although most scan operations have variants that use default values.

While these variables are passed in during operation startup, you can also update them while the scan is running. This enables you, for example, to reduce impact during working hours, while allowing more resources to be used during off hours.

Since the scan is running, of course, you need to update these variables from a second session, using the `sessionId` of the session that is running the scan.

One way to determine the session Id of the session that is running a scan operation is by checking the session holding the `GcLock`. However, while only one session can be holding the `GcLock` at a time, and `markForCollection` requires the `GcLock`, other operations such as `GsObjectInventory`, also may have the `GcLock`.

To access the upper limit on the number of threads:

```
System mtThreadsLimit: aSessionId
```

To update the upper limit on the number of threads:

```
System mtThreadsLimit: aSessionId setValue: anInt
```

To access the CPU load limit:

```
System mtPercentCpuActiveLimit: aSessionId
```

To update the CPU load limit:

```
System mtPercentCpuActiveLimit: aSessionId setValue: anInt
```

Both of these variables are used in tuning, but they have somewhat different uses. The primary way you will tune the impact on your system is by setting `MtPercentCpuActiveLimit`. The operation then controls its impact by activating or deactivating threads, up to a limit of `MtThreadsLimit`. The operation will proceed, using more or less resources at any particular time depending on what else is executing on your system. Note that the CPU load includes non-GemStone process running on this same machine, so if a machine is heavily used by non-GemStone processes, the operation may make little progress even if the GemStone repository itself is idle.

`MtThreadsLimit` acts as a ceiling on the impact as well. Since this limit is of more relevance within GemStone, on heavily loaded machines you may want to pay more attention to this limit to control the impact within the repository. This limit is also useful when you want to pause the scan. Setting the `MtThreadsLimit` to 0 means that the scan cannot perform work, but does not stop executing, it waits until a non-zero limit is set.

Cache Statistics

The following cache statistics are important for tuning multi-threaded scans. These are visible by using statmonitor data viewed in VSD (the visual statistics display tool); see *VSD User's Guide*. You may also access methods in System to get the values programmatically; see "Programmatic Access to Cache Statistics" on page 138 for more information.

<code>MtThreadsLimit</code>	The upper limit on the number of threads that can be running at any one time.
<code>MtPercentCpuActiveLimit</code>	The upper limit on percent of CPU that can be active before threads are deactivated.
<code>percentCpuActive</code>	The current percentage of CPU that is active.
<code>MtActiveThreads</code>	The current number of active threads

Memory Impact

Multi-threaded operations may require considerable heap memory. This memory requirement is **not** part of temporary object cache memory. You can configure your `GEM_TEMPOBJ_CACHE_SIZE` according to other application Gem requirements, or even configure the sessions performing repository scan operations with a very small temporary object cache size.

The amount of memory space that is needed depends primarily upon the current `oopHighWater` value, the number of threads, and the page buffer size. `markForCollection` uses a `pageBufferSize` of 128, `epoch` and `writeSetUnionSweep` use a size of 64, and it is an explicit argument to `FDC`.

The overhead associated with the `oopHighWater` value can be computed as:

$$(\text{stnOopHighWater} + 10\text{M}) / 2$$

The memory cost per thread is:

$$50\text{K} + (180\text{K} * \text{pageBufSize})$$

For example, a system with an `oopHighWater` mark of 500M running eight threads with a page buffer size of 128 would require a minimum of about 440 MB of free memory.

Identifying Sessions Holding Up Voting

Voting is the 4th phase of garbage collection, described in Step 4 on page 260. During this phase, each logged-in gem must vote on possibly dead objects. Sessions perform this vote on the next abort or commit that they execute, or on logout. If there are idle sessions that do not commit or abort, voting will not be able to complete.

You may find these sessions using:

```
System class >> notVotedSessionNames
```

The method `System class >> descriptionOfSession:` can help in tracking down such sessions. The array returned by this method includes the not voted status, as element 20. For details, see the comment in the image.

Tuning Write Set Union Sweep

The write set union sweep is the 5th phase of garbage collection, described in Step 5 on page 261. It is performed by the Admin Gem.

The write set union sweep is performed using the Multi-Threaded Scan (page 278), and can be tuned using the following `GcGem` parameters:

<code>#sweepWsUnionPercentCpuActiveLimit</code>	Limit active <code>wsUnion</code> threads when system <code>percentCpuActive</code> is above this limit. Default: 90.
<code>#sweepWsUnionPageBufferSize</code>	Size (in pages) of buffer used for <code>wsUnion</code> sweep. Must be a power of 2. Default: 64. Minimum: 8. Maximum: 1024.
<code>#sweepWsUnionMaxThreads</code>	The maximum threads used for next <code>wsUnion</code> sweep. By default, use one thread.

Identifying Sessions Holding Up Page Reclaim

Reclaiming pages can proceed only up to those pages currently providing some session's transaction view of the repository – that is, only up to the oldest commit record. When other sessions are logged in, reclaim stops at that point until all sessions using that commit record either commit or abort their transaction.

It can be helpful to identify which sessions are holding on to the oldest commit record. The method `System class>>sessionsReferencingOldestCr` returns an array of session IDs, which can be mapped to GemStone logins through

System class>>currentSessionNames or
 System class>>descriptionOfSession:*aSessionId*. For example:

```
topaz 1> printit
System sessionsReferencingOldestCr
%
an Array
#1 5

topaz 1> printit
System currentSessionNames
%
session number: 2      UserId: GcUser
session number: 3      UserId: GcUser
session number: 4      UserId: SymbolUser
session number: 5      UserId: DataCurator
```

The method `descriptionOfSession:` is particularly useful in that it returns an array of descriptive information. The second element is the operating system process ID (pid), and the third element is the name of the node on which the process is running. For details, see the comment in the image.

Finding large objects that are using excessive space

If you know that you have large objects that are no longer needed, another way to free space is to explicitly remove references to them. To remove such objects, you must first identify them. Then you can find all references to them and remove those references.

Identify Larger Objects in the Repository

The following methods returns all objects that are over a specified size:

```
Repository >> allObjectsLargerThan: aSize
Repository >> fastAllObjectsLargerThan: aSize
```

These methods return a `GsBitmap` of all objects in the repository larger than *aSize*; objects for which you do not have read authorization are not included.

You can convert the `GsBitmap` to an `Array` using `asArray`, or retrieve only a specified number of results using `removeCount:` (which removes elements from the bitmap) or `peekCount:` (which does not remove them).

This method performs the repository wide search using the Multi-Threaded Scan (page 278). `allObjectsLargerThan:` uses two threads and a CPU limit of 100%, so it may have some impact, particularly on smaller host systems, and may take some time to complete. To complete the operation as quickly as possible, using as much of the system resources as necessary, you can use the `fast` version..

Finding named objects that are large

Named objects are Global variables; objects that have a reference by a `Symbol` name in some user's `SymbolDictionary`. While there are some legitimate uses of Globals for environment-wide information, generally using global variables (other than for classes) is not good software engineering practice.

A common pattern is to use a global to keep some objects persistent, such as an expression:

```
UserGlobals at: #tempCollection put: IdentityBag new
```

If this collection and its contents is not deleted, it may continue to use space and may not be easily noticed. Using SessionTemps is preferred to avoid this problem.

The following expression causes GemStone to look through the symbol list for each user in AllUsers and gather information on any named objects larger than the SmallInteger *aSize*. Since it is looking for named references, it does not need to do a repository scan.

```
topaz 1> printit
AllUsers findObjectsLargerThan: aSize limit: aSmallInt
%
```

This method locates large collections or strings referenced by name; it will not locate collections stored within the class variables of classes, or in instances of classes. It returns an Array of up to *aSmallInt* elements, each of the form { { *aUserId* . *aKey* . *anObject* } }, where *anObject* is an object larger than *aSize* defined in the symbol list of *aUserId*, and *aKey* is the Symbol associated with that object.

Finding References to an Object that prevent garbage collection

Full reference path

To find a complete reference path to a particular object, you can use methods on the GsSingleRefPathFinder. This allows you to determine the complete reference path from a root object to the argument.

NOTE

This method runs in transaction, and may take a considerable time to run. Avoid using it in production systems.

To perform the scan, you create an instance of the GsSingleRefPathFinder for the object or objects, run the scan, and collect/view the results.

Steps to find a reference path:

Step 1. Create an instance with default settings:

```
inst := GsSingleRefPathFinder newForSearchObjects:
(Array with: mySearchObject).
```

Step 2. Run the scan

```
inst runScan.
```

Step 3. Build the result

```
resultObjs := inst buildResultObjects.
```

`buildResultObjects` returns a collection of instance of GsSingleRefPathResult, which is a subclass of Array. Each element in the GsSingleRefPathResult represents an element in the reference path; it also has instance variables for the searchOop and status.

Step 4. 4) For display, collect the results as strings:

```
resultObjs collect: [:e | e resultString].
```

Steps 2-4 can be done using the `GsSingleRefPathFinder` method `scanAndReport`. For example:

```
(GsSingleRefPathFinder newForSearchObjects: { mySearchObject })
    scanAndReport
```

These methods locate the first path from a root object to the argument object that is found, but there may be multiple paths.

Note that you cannot find references to a Class or Metaclass using these methods.

Once you have found the references to the unwanted object, set those references to `nil`. This allows the object to be removed during normal garbage collection.

Example 14.1 Finding reference path

```
| inst |
inst := GsSingleRefPathFinder newForSearchObjects:
    { PlusInfinity }.
inst runScan.
(inst buildResultObjects)
    collect: [:e | e resultString]
%
anArray( 'Reference path for search oop 21393665 (Float)
1 207361 (SymbolDictionary)
2 1126401 (IdentityCollisionBucket)
3 1790721 (SymbolAssociation)
4 21393665 (Float)
')
```

In this example, `PlusInfinity` is a `Float` (#4). It is referred to in a `SymbolDictionary` (#1), using internal implementation objects (an `IdentityCollisionBucket` and a `SymbolAssociation`) that actually have the references.

`GsSingleRefPathFinder` provides a number of instance variables to parameterize the search:

- ▶ `maxThreads`
- ▶ `lockWaitTime`
- ▶ `pageBufferSize`
- ▶ `percentCpuLimit`
- ▶ `maxLimitSetDescendantObjs`
- ▶ `maxLimitSetDescendantLevels`
- ▶ `printToLog`

Defaults are provided, or you may send messages, for example to perform a less aggressive scan.

All References

You can search the repository for multiple references to an object by sending the following message:

```
topaz 1> printit  
anObject findReferencesWithLimit: aSmallInt  
%
```

This returns an Array of objects in the repository that reference *anObject*. The results will not include duplicates, if an object contains multiple references to *anObject*. The Array is limited to a maximum of *aSmallInt* elements, or use a negative to get unlimited results.

The resulting Array contains only references associated with *GsObjectSecurityPolicies* for which you have read authorization. If any references to *anObject* are protected by *GsObjectSecurityPolicies* for which you do not have read authorization, the last element of the result is the String 'Read Authorization Error encountered'.

findReferencesWithLimit: uses the Multi-Threaded Scan (page 278). It uses a default of two threads and up to 90% of the CPU.

GemStone Configuration Options

This appendix describes the GemStone/S 64 Bit configuration settings that control many facets of the GemStone system. It covers:

How GemStone Uses Configuration Files (page 285)

describes system-wide and executable-dependent configuration files, how the system locates configuration files, and how to setup customized configuration files

Configuration File Syntax (page 291)

lists syntax used within configuration files.

Configuration Options (page 294)

describes of all options that can be used within configuration files

Runtime-only Configuration Options (page 332)

lists configurations options that do not have settings for a configuration file, but have runtime settings that can be used in a running system.

A.1 How GemStone Uses Configuration Files

A GemStone/S 64 Bit configuration file is a file containing information that, when read at startup time, can control the configuration, behavior, and functionality of the system at run time. Some of these configuration settings can be modified dynamically by sending messages in GemStone Smalltalk.

The Stone, Gem, and linked applications (collectively, the repository executables) are able to read two different types of configuration files: system-wide configuration files and executable-dependent configuration files.

- ▶ **System-wide configuration files** allow the GemStone system administrator to set options pertaining to all GemStone executables on a system- or network-wide basis. This file is required for a Stone to start.

System-wide configuration files are found in a default location, passed by the `-z` argument, or located by a `GEMSTONE_SYS_CONF` environment variable.

- ▶ **Executable-dependent configuration files** can be used by individual users to control their own running copy of the GemStone system. Options contained in executable-dependent configuration files override the options specified in a system-wide configuration file.

Executable-dependent configuration files are found by name, passed by the `-e` argument, or located by a `GEMSTONE_EXE_CONF` environment variable.

These environment variables can be set in the usual way. For example:

```
$ GEMSTONE_EXE_CONF=$HOME/myFile.conf
$ export GEMSTONE_EXE_CONF
```

Both `GEMSTONE_SYS_CONF` and `GEMSTONE_EXE_CONF` can be defined to point to either a file or a directory.

At startup time, GemStone repository executables attempt to find and read both a system-wide and an executable-dependent configuration file, searching for these files in the following manner.

System Configuration File

GemStone repository executables begin by attempting to find a system-wide configuration file.

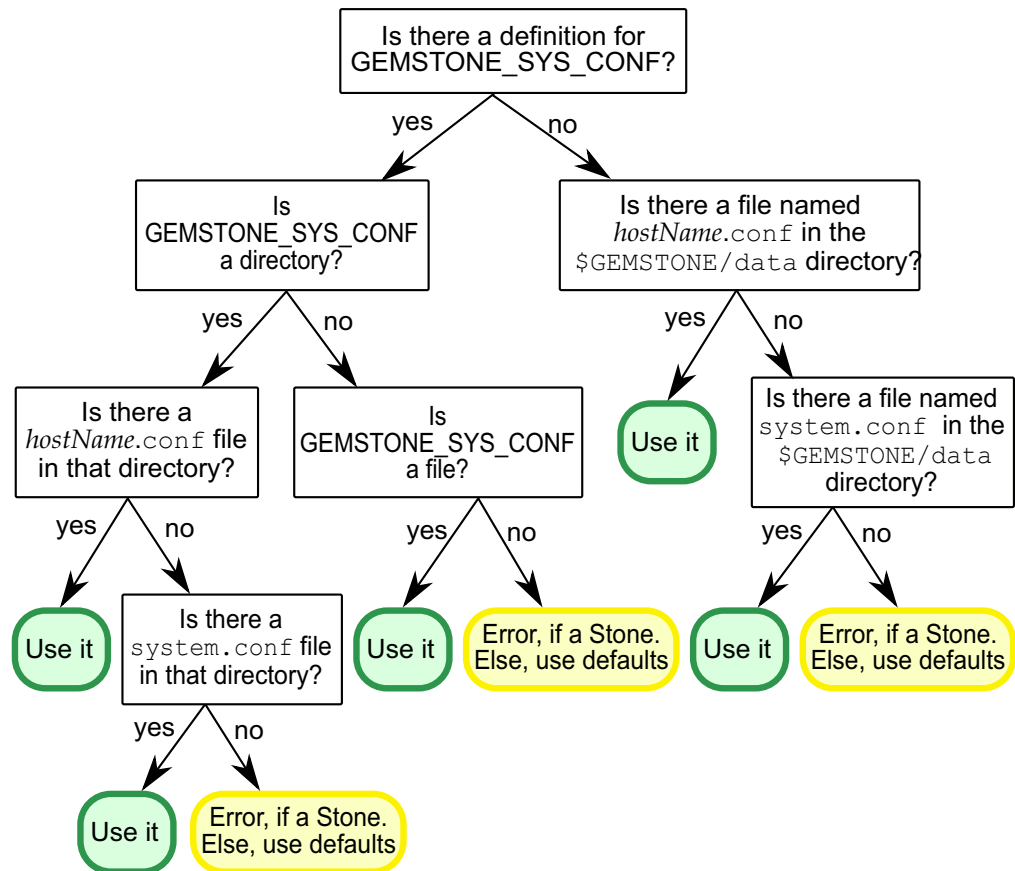
1. As shown in Figure A.1, GemStone first checks to see if there is an environment variable defined for `GEMSTONE_SYS_CONF`.
2. If `GEMSTONE_SYS_CONF` is *not* defined, GemStone looks for a file named `hostName.conf` in `$GEMSTONE/data` and uses that file. `hostName` must match the results of executing the `hostname` command on the machine on which the executables are running.
3. If no such file exists, it looks for a file named `system.conf` in `$GEMSTONE/data` and uses that.
4. If neither of those files exist, the system defaults are used, unless the executable is Stone, in which case an error is generated.
5. If `GEMSTONE_SYS_CONF` is defined, GemStone checks to see if it points to a directory.
6. If `GEMSTONE_SYS_CONF` points to a directory, GemStone looks for a file named `hostName.conf` in that directory. If it finds such a file, it uses it. `hostName` must match the results of executing the `hostname` command on the machine on which the executables are running. If no `hostName.conf` is found, it looks in that directory for a file named `system.conf` and uses that. If neither of those files exist, the system defaults are used, unless the executable is Stone, in which case an error is generated.

If the `GEMSTONE_SYS_CONF` environment variable points to a file instead of a directory, GemStone just uses that file.

Within each file, if an option is listed more than once, then the value it is given the last time it is specified is used as its true value at executable run time.

This rule also applies between the two types of configuration files. If the same option is given a value in both the system and executable configuration files, the value in the executable configuration file overrides the system configuration file's value.

Figure A.1 Search Path for a System Configuration File

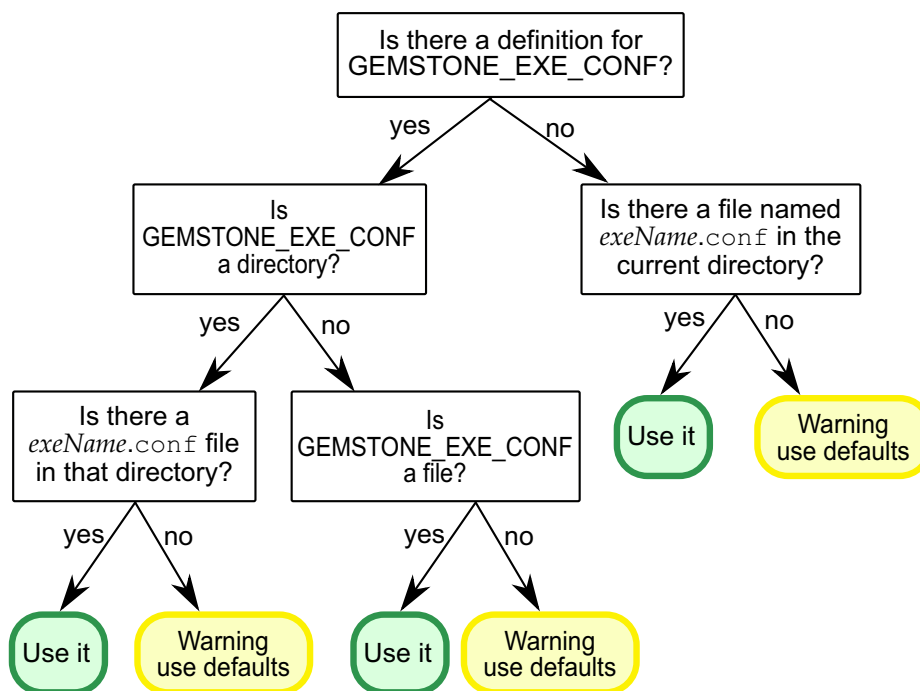


Executable Configuration File

Ordinarily, GemStone repository executables next try to find an executable-dependent configuration file. (The exception is a Stone repository monitor that failed to find its system-wide configuration file – it exits with an error.)

1. As shown in Figure A.2, GemStone begins by checking to see if there is an environment variable defined for `GEMSTONE_EXE_CONF`.
2. If `GEMSTONE_EXE_CONF` is not defined, GemStone tries to find a file called `exeName.conf` in the current working directory. (For information about the naming conventions, see “Naming Executable Configuration Files” on page 289.)
3. If it succeeds at finding such a file, it uses that file. If such a file does not exist, it generates a warning and relies solely on the system configuration file for configuration parameters.

Figure A.2 Search Path for an Executable Configuration File



If `GEMSTONE_EXE_CONF` is defined, GemStone first looks to see if it points to a directory.

- ▶ If `GEMSTONE_EXE_CONF` points to a directory, GemStone looks for a file named `exeName.conf` in that directory. If such a file exists, it uses it; if not, a warning is generated and GemStone relies on the system configuration file for configuration parameters.
- ▶ If `GEMSTONE_EXE_CONF` points to a file, rather than to a directory, GemStone simply uses that file.
- ▶ If `GEMSTONE_EXE_CONF` points to a directory or file that doesn't exist, a warning is generated and GemStone defaults to using the system configuration file for configuration parameters.

Creating or Using a System Configuration File

If you are satisfied with the standard options and the defaults, the simplest thing to do is to just use the configuration file provided in `$GEMSTONE/data/system.conf`. You can either copy this file and set the `GEMSTONE_SYS_CONF` environment variable to point to your new file, or you can do nothing and let GemStone use `$GEMSTONE/data/system.conf` itself.

Creating an Executable Configuration File

There are two ways to create a configuration file for a specific executable:

- ▶ You can copy the entire system-wide configuration file to a new file, name it appropriately, and change selected parameters.

- ▶ You can create a new file, give it an appropriate name, and include only those parameters that you want to differ from the default.

To make sure that GemStone is able to find and use your executable configuration file, you can set the `GEMSTONE_EXE_CONF` environment variable to point to your file. `GEMSTONE_EXE_CONF` can be either a file name or a directory name. If you set the environment variable to a directory name, be sure to name the configuration file `exeName.conf` so GemStone can find it at start up. (Information about the naming conventions for configuration files is just ahead.)

If you don't set the `GEMSTONE_EXE_CONF` environment variable, GemStone looks for a file named `exeName.conf` in the current working directory at startup. If it doesn't find one, it uses the configuration parameters set in the system configuration file, or it uses the system defaults.

NOTE

Make sure your executable-dependent file is both readable and writable by the Stone process, which will update options by writing to it if you make certain configuration changes at run time.

Nesting Configuration Files

Configuration files can include other configuration files using the option `INCLUDE`. This may be a more intuitive way to manage multiple configuration settings that are shared and unshared among individual clients.

The `INCLUDE` configuration parameter can be set to the name of a single file, which will be read, along with the configuration file itself, when a process starts. Unlike other parameters, more than one `INCLUDE` may be included in a configuration file, and each included file is read, not just the last one. Nesting may be up to 100 levels.

When a duplicate parameter definition is found, the last one is used. Location of the `INCLUDE` directive determines the last value; if the `INCLUDE` directive is at the end of the file, values in the nested configuration file override the current file values.

For example:

```
INCLUDE=config2.conf;  
INCLUDE="$GEMSTONE/data/config3.conf";
```

Naming Executable Configuration Files

The default name of an executable configuration file generally is determined from the name of the executable itself.

Application Gems

Stand-alone (RPC) Gems look for a file named `gem.conf` in the current working directory unless `GEMSTONE_EXE_CONF` is defined. The working directory by default is the user's home directory, unless otherwise specified by the NRS or a customized `gemnetobject` script.

The file `$GEMSTONE/sys/gemnetobject` is a script that a NetLDI invokes to start a GemStone session process. This script can be edited to define the name of the Gem to execute, the directory where the Gem resides, and the `GEMSTONE_SYS_CONF` and `GEMSTONE_EXE_CONF` environment variables.

The NetLDI that is specified by the login parameters will lookup tries to map the requested object to the path of an executable by looking for an entry in `$GEMSTONE/sys/services.dat`. That file contains an entry for the standard Gem session process:

```
gemnetobject          $GEMSTONE/sys/gemnetobject
```

For example, when you enter "gemnetobject" as a session login parameter (such as for gemnetid in Topaz), the NetLDI uses the `services.dat` file to map the request to the script `$GEMSTONE/sys/gemnetobject`. Similarly, an object name can be entered while setting up a GemBuilder session (as Name of Gem Service) or other application.

Application programmers provide the name as a parameter to `GciSetNet()`.

If the NetLDI does not find the requested object in `services.dat`, it searches for an executable with that name in the user's `$HOME` directory. If you have a private Gem executable, place the executable in `$HOME` and then enter its name in place of `gemnetobject` during a GemStone login. Because of the search order, the private name must not be the same as that of an object in `services.dat`. The name must be the name of a file in `$HOME`, not a pathname.

System Gems

It is sometimes useful to change the parameters in a configuration file specific to a system Gem, such as the Admin or Reclaim Gem. This allows customized configuration settings that remain in effect if the system is stopped and restarted.

To do so:

Step 1. Create a text file with the configuration settings you want.

Step 2. Save your changes with an appropriate filename (for example, `adminmgcgem.conf`, `reclaimmgcgem.conf`, etc., depending on which system Gem the new configuration file is for). Place the file in GemStone's `sys` directory.

Step 3. Make a copy of the appropriate script/s.

```
runadminmgcgem      Starts the Admin Gem.
runcachewarmergem  Starts the cache warmer Gems.
runotcachewarmergem Starts the Object Table cache warmer Gems.
runreclaimmgcgem   Starts the Reclaim Gem.
runsymbolgem       Starts the SymbolGem.
```

These scripts are located in `$GEMSTONE/sys/`. Name the copy appropriately; for example, `$GEMSTONE/sys/myrunreclaimmgcgem`

Step 4. Edit `myrunreclaimmgcgem` to specify the customized configuration file.

For example, to use a Reclaim Gem configuration file named `$GEMSTONE/sys/reclaimmgcgem.conf`, locate the lines:

```
exeConfig=" "
```

and change to:

```
exeConfig="$GEMSTONE/sys/reclaimmgcgem.conf"
```

Step 5. Edit the file `$GEMSTONE/sys/services.dat`. Comment out the existing line:

```
runreclaimmgcgem      $GEMSTONE/sys/runreclaimmgcgem
```

and add an entry specifying the new script to be executed for the `runreclaimcgem` service.

```
#runreclaimcgem      $GEMSTONE/sys/runreclaimcgem
runreclaimcgem      $GEMSTONE/sys/myrunreclaimcgem
```

Stone

Stone looks for a file named `stoneName.conf` in the current working directory.

Linked Topaz

The linked version of Topaz looks for the configuration file `gem.conf` in the current working directory, so, by default, Gem and Topaz can share the same options.

Linked GemBuilder for Smalltalk (GBS)

Linked GBS logins by default look for a file named `gbs.conf` in the current working directory.

Linkable GemBuilder for C Applications

Linkable GemBuilder for C applications look for a file named `gci.conf` in the current working directory unless the application has provided a different name by calling `GciInitAppName()`.

Naming Conventions for Configuration Options

The prefix “GEM_” indicates that the option is processed directly by Gems. Unless indicated otherwise by the phrase “used by all executables,” most other options are processed only by the Stone, which passes the information to executables as needed through network connections. Exceptions are the shared page cache configuration options (“SHR_”). The first Gem session process on a node remote from the Stone and extents reads these options, which determine the configuration of the shared page cache on that node.

All executables (that is, the Stone and Gems) understand the standard options used in the file `$GEMSTONE/data/system.conf` as shipped. The GemStone executables generate a warning message whenever they encounter an option that is not in the standard list.

NOTE:

If the DUMP_OPTIONS option is set to true, then once the system-wide and executable-dependent configuration files have been processed, the values of all the options understood by the executable are displayed. You can access the configuration parameters from Smalltalk by using the methods described starting on page 55.

A.2 Configuration File Syntax

The following section describes the rules of grammar to be used in editing configuration files.

White space

Leading white space and white space before and after the equal and semicolon termination symbol are ignored.

New lines

New lines within a statement are allowed only after an equal sign or after a comma within a list of values.

Comments

The comment symbol for GemStone configuration files is the pound sign (#). Any text following the pound sign in a line is ignored.

Lists

Lists are separated by commas; list elements can be empty, for example:

```
DBF_EXTENT_SIZES = 2GB, , 2GB;
```

Within lists of values, leading and trailing white space is ignored.

Strings

Strings may be encased in single or double quotes, or if no spaces or escape are included, the quotes can be omitted. An empty string is acceptable, and may be expressed using quotes (""), or by no value at all (for instance, OPTION = ;).

Case-sensitivity

String option values are case-sensitive; boolean option names are *not* case-sensitive.

Maximum Sizes

The maximum length of a string option is 1024 characters. There is no limit on the number of elements within a list.

Use of Environment Variables In Options

Options that are either file names or directories may have environment variables as the first part of their value or the entire value.

Errors in Configuration Files

At startup, each GemStone executable reads the configuration files. If any error is detected, information about the error is written to the standard output. This information indicates the file and line containing the error and the error's severity.

If the CONFIG_WARNINGS_FATAL configuration option (page 294) is set to true, then any errors in the configuration file/s read by the process will cause the process to terminate. This avoids inadvertently getting default configuration values. When CONFIG_WARNINGS_FATAL is false, some invalid settings will still result in execution

Two kinds of errors can be generated by the processing of configuration files: syntax errors and option value errors.

Syntax Errors

Syntax errors are generated whenever a grammatical error is detected in the configuration file. All syntax errors are warnings; they do not cause execution to terminate unless CONFIG_WARNINGS_FATAL is true. These errors include:

- ▶ End-of-line or end-of-file detected before expected
- ▶ Invalid starting character for an option name or invalid character within an option name
- ▶ Equals or semicolon sign expected
- ▶ Invalid 3-digit escape sequence

- ▶ Invalid escape character
- ▶ Terminating quote missing in a quoted string

Option Value Errors

Option value errors are generated when the value assigned to an option has no meaning or is of the wrong type. For example, an option value error is generated when an option defined to need a boolean for its value has been set to an integer.

Option value errors vary in severity. Some options, such as not specifying the list of files that make up a logical repository, will necessarily terminate execution. Other option value errors, such as a invalid cache size, might only generate warnings, unless CONFIG_WARNINGS_FATAL is true. When a warning is issued and CONFIG_WARNINGS_FATAL is false, the executable ignores the given value and use the option's default value.

Alternate ways to specify configuration parameter values

Usually, configuration parameter values are specified in configuration files. These are found according the rules described earlier using the environment variables, default locations, or by values passed to the -e and -z arguments for GemStone utilities that accept these arguments (such as Stone and Topaz).

Configuration parameter values may also be passed in to Gem and Linked sessions using the -C argument. This is accepted by the Gem NRS and by topaz. The argument string must follow the configuration file syntax rules, and must be quoted, double quoted, and escaped as necessary.

For example, to set a configuration parameter for a RPC Gem, in Topaz:

```
set gemnetid 'gemnetobject -C GEM_TEMPOBJ_OOMSTATS_CSV=TRUE'
```

To set a configuration parameter in linked Topaz:

```
topaz -l -C GEM_TEMPOBJ_OOMSTATS_CSV=TRUE
```

A.3 Configuration Options

The system configuration file contains the following standard configuration options. In this discussion, *default* refers to the value that results when an option is not explicitly set by a statement in the configuration file. *Initial setting* refers to an explicit setting in the initial `system.conf` file that differs from the default.

Some configuration options have an internal runtime parameter that can be changed while GemStone is running. Where such a parameter exists, its name is given as part of the entry. For more information, see “To Change Settings at Run Time” on page 56.

The `$GEMSTONE/bin` directory contains a write-protected file named `initial.config` that is an exact replicate of `$GEMSTONE/data/system.conf` as it was originally shipped. After modifying `system.conf`, you can always recover its original condition.

CONFIG_WARNINGS_FATAL

If `CONFIG_WARNINGS_FATAL` is set to `TRUE`, then any warnings about invalid or out of range entries in a configuration file are treated as fatal errors, terminating the Gem or Stone process that is reading the configuration file. The last occurrence of this parameter in any configuration file (in the case of multiple settings within a file, or settings in nested configuration files), will control the value used. If the last occurrence is `TRUE`, then any warnings from preceding configuration parameter will be treated as fatal.

Default: `FALSE`

DBF_ALLOCATION_MODE

`DBF_ALLOCATION_MODE` describes the space allocation heuristic to be used when filling repository extents.

Permissible values are either `Sequential` or a series of allocation weights, separated by commas. Under sequential allocation, each extent has its full resources used before the next extent's resources are used. Under weighted allocation, those extents with a larger weight will have proportionally more of their disk resources allocated than those with smaller weights. Each weight applies to the corresponding extent in the series of extents specified in `DBF_EXTENT_NAMES`, and the number of elements must match. Extent allocation weights must be integers in the range 1..40 (inclusive).

Default: `Sequential`

DBF_EXTENT_NAMES

`DBF_EXTENT_NAMES` list of all repository extents, in order, primary extent first, separated by commas. Taken together, all of the listed file resources make up the logical repository. This option is required, and must contain at least one entry, the name of the primary extent. The maximum number of extents is 255.

An extent name can be a file name or the device name for a raw disk partition. The name can have an environment variable as its first component.

Default: `EMPTY`. The system will not run unless you define an extent list.

Initial setting: `$GEMSTONE/data/extent0.dbf`

DBF_EXTENT_SIZES

DBF_EXTENT_SIZES sets the maximum sizes of all repository extents, in order, primary extent first, separated by commas. Each size applies to the corresponding extent in the series of extents specified in DBF_EXTENT_NAMES.

A size entry may be empty, which indicates that the corresponding extent has no fixed maximum size. This setting allows the extent to grow until it fills the disk containing it.

The actual maximum size of an extent will always be a multiple of 16 MB. If an extent size specified by DBF_EXTENT_SIZES is not a multiple of 16 MB, then the actual maximum size will be the next lowest multiple of 16 MB. For example, an extent created with a maximum size of 260 MB, this extent really has a maximum size of only 256 MB.

When an extent is on a raw partition, for optimal performance the corresponding setting in DBF_EXTENT_SIZES should be 16MB smaller than the size of the partition. For example, set it to about 1984 MB for a 2 GB partition.

You can modify the size of an existing extent under these conditions:

- ▶ If the original maximum size was unlimited, the new maximum size must be larger than the current physical size of the extent.
- ▶ If the original maximum size was limited, the new maximum size must be larger than the original maximum size.

The Stone repository monitor is the only executable allowed to change DBF_EXTENT_SIZES. At GemStone system startup, the maximum size of each extent is written to the system log.

If no units are specified, the value is in MB (1 Megabyte = 1048576 bytes). You may also specify units as KB, MB, or GB.

Default Units: MB

Min: 16 MB

Max: 33554432 MB (subject to disk, operating system and platform limits)

Default: EMPTY (no maximum sizes)

DBF_PRE_GROW

If DBF_PRE_GROW is set to TRUE and there are extents for which a size is specified in DBF_EXTENT_SIZES, then on repository startup, each extent with a size in DBF_EXTENT_SIZES larger than the current size will be pregrown to the specified size. If the grow fails, the extent is reset to its original size and startup fails.

If DBF_PRE_GROW is set to TRUE and a new extent is added programmatically with a size specified, it will be pregrown to that size. If the extent cannot be grown to the maximum size because of disk capacity problems, then extent creation will fail.

The default value for DBF_PRE_GROW is FALSE. This setting indicates that extents will grow only when new space is needed. An extent without a maximum size is never pregrown.

The value of DBF_PRE_GROW may also be a list of integer sizes, which may include blanks for specific extents that will not be pregrown. Extents with an integer size specified in DBF_PRE_GROW will be pregrown to this size if needed. Extents without an entry in DBF_PRE_GROW will not be pregrown.

It is an error if the value for an extent's DBF_PRE_GROW size is larger than the corresponding DBF_EXTENT_SIZES size; if one but not both are empty, it is not an error and the extent will not be pregrown.

Elements of DBF_PRE_GROW may be blank to specify pregrow sizes for some but not all extents, such as:

```
DBF_PRE_GROW = 1000, , 1000;
```

If no units are specified, the value is in MB (1 Megabyte = 1048576 bytes). You may also specify units as KB, MB, or GB.

Default Units: MB

Min: 1 MB

Max: 33554432 MB (limited by DBF_EXTENT_SIZES values)

Default: FALSE

DBF_SCRATCH_DIR

DBF_SCRATCH_DIR specifies a scratch directory that the Stone process can use to create “scratch” repositories for use during **pageaudit**. The file name is appended to the directory name *without* an intervening delimiter, so a trailing delimiter is necessary here.

Default: \$GEMSTONE/data/

DUMP_OPTIONS

If DUMP_OPTIONS is set to true, dumps a summary of all configuration options as part of the process log file headers.

Default: true

GEM_ABORT_MAX_CRIS

When a Gem is not in a transaction and aborts, GEM_ABORT_MAX_CRIS specifies the maximum number of commit records to analyze to compute the writeSetUnion since the last time this session aborted. If the number of commit records would exceed this limit, the abort is treated similar to a LostOt and all in-memory copies of committed objects are marked invalid and will be re-read as needed during subsequent execution.

A value of 0 (zero) means no limit on number of commit records to analyze.

Runtime parameter: **#GemAbortMaxCris**

Min: 0

Max: 2147483647

Default: 0

GEM_CACHE_WARMER_ARGS

Configures cache warming on a remote cache. If it is an empty string (the default), no cache warming is done when the remote gem starts a remote shared page cache. If it is set to a string containing spaces, a default cache warmer is started.

Otherwise, the value should contain valid startcachewarmer options and values that will be used to invoke the cache warmer on the remote cache.

The arguments used here will use to fork the startcachewarmer utility and have the same meanings. Not all startcachewarmer arguments are valid: only: -d -D -l -L -n -w. For the meanings and details for these arguments, see “startcachewarmer” on page 347.

This option is only used by remote gems that create a remote shared page cache. It is ignored by all other gems.

For example, to start a cache warmer with 5 threads and write the working set on cache shutdown:

```
GEM_CACHE_WARMER_ARGS = "-n 5 -w 0";
```

Default: "" (cache warmer will not be started)

GEM_CACHE_WARMER_MID_CACHE_ARGS

Configures cache warming on a mid-level cache. If it is an empty string (the default), no cache warming is done when the remote gem starts a mid-level shared page cache. If it is set to a string containing spaces, a default cache warmer is started.

Otherwise, the value should contain valid startcachewarmer options and values that will be used to invoke the cache warmer on the mid-level cache.

The arguments used here will be passed to the startcachewarmer utility and have the same meanings. Not all startcachewarmer arguments are valid: only: -d -D -l -L -n -w. For the meanings and details for these arguments, see “startcachewarmer” on page 347.

For example, to start a mid-level cache warmer with 5 threads and write the working set on cache shutdown:

```
GEM_CACHE_WARMER_MID_CACHE_ARGS = "-n 5 -w 0";
```

Default: "" (cache warmer is not started)

GEM_COMPRESS_TRANLOG_RECORDS

If true, sessions will compress tranlog data records using lz4 compression before sending them to stone .

Default: true

GEM_FREE_FRAME_CACHE_SIZE

GEM_FREE_FRAME_CACHE_SIZE specifies the size of the Gem’s free frame cache. When using the free frame cache, the Gem removes enough frames from the free frame list to refill the cache in a single operation. When adding frames to the free list, the Gem does not add them until the cache is full.

A value of 0 disables the free frame cache (the Gem acquires frames one at a time). A value of -1 means use the default value: 0 for caches less than 100 MB and 10 for caches of 100 MB or greater.

Cache Statistic: **FreeFrameCacheSize** (Gem)

Units: frames

Min: -1

Max: 63

Default: -1 (see above discussion)

GEM_FREE_FRAME_LIMIT

When the number of free frames in the shared page cache is less than GEM_FREE_FRAME_LIMIT, the Gem session process scans the cache for a free frame rather than using one from the free frame list. This action is desirable for performance

reasons so the remaining frames in the list are available for use by the Stone repository monitor.

If the value of `GEM_FREE_FRAME_LIMIT` is `-1`, the free frame limit is set to one of the following default values:

- ▶ For primary shared page cache that is 800 MB or smaller: 10% of the number of frames in the cache
- ▶ For primary shared page cache greater than 800 MB: 5000 frames
- ▶ For a remote shared page cache: 0

Runtime parameter: **#GemFreeFrameLimit**

Cache Statistic: **FreeFrameLimit** (Gem)

Min: 1

Max: 65536

Default: `-1` (see above discussion)

GEM_FREE_PAGEIDS_CACHE

`GEM_FREE_PAGEIDS_CACHE` specifies the maximum number of free pageIds to be cached in Gem. Larger values reduce number of calls to Stone, at a cost of needing more free space within the extents.

Runtime parameter: **#GemFreePageIdsCache**

Min: 40

Max: 3500

Default: 200

GEM_HALT_ON_ERROR

`GEM_HALT_ON_ERROR` causes a Gem to halt and dump core if an error with the specified GemStone error number occurs. The value `-1` means “never halt”. Ordinarily this option is used only to assist Technical Support in diagnosing problems.

Runtime parameter: **#GemHaltOnError**

Default: `-1`

GEM_KEEP_MIN_SOFTREFS

`GEM_KEEP_MIN_SOFTREFS` determines the minimum number of most recently used SoftReferences that will not be cleared by VM markSweep if *startingMemUsed* – the percentage of temporary object memory in use at the beginning of a VM mark/sweep – is greater than `GEM_SOFTREF_CLEANUP_PERCENT_MEM` but less than 80%.

In most cases, the default (0) is appropriate and should not be changed.

Runtime parameter: **#GemKeepMinSoftRefs**

Min: 0

Max: 10000000

Default: 0

GEM_KERBEROS_KEYTAB_FILE

Path to the Kerberos key table file. The keytab file only is required when passwordless logins to GemStone are in use. The file contains pairs of Kerberos principals and encrypted keys. In this case, the Kerberos service is the service for a GemStone repository.

Refer to the GemStone System Administration Guide for information on creating and maintaining this file.

Default: NONE

GEM_KEYRING_DIRS

A list of directories which contain keys and certificates used for secure backup and restore.

Runtime equivalent: `#GemKeyRingDirs`

Default: NONE

GEM_MAX_SMALLTALK_STACK_DEPTH

`GEM_MAX_SMALLTALK_STACK_DEPTH` determines the size of the GemStone Smalltalk execution stack space that is allocated when the Gem logs in. The unit is the approximate number of method activations in the stack. This setting causes heap memory allocation of approximately 64 bytes per activation. Exceeding the stack depth results in generation of the error `RT_ERR_STACK_LIMIT`.

Min: 100

Max: 1000000

Default: 1000

GEM_NATIVE_CODE_ENABLED

`GEM_NATIVE_CODE_ENABLED` enables or disables generation of native code. This is set to an integer 0, 1, or 2. For compatibility with configuration files from earlier versions, it may be also set to `TRUE` or `FALSE`.

Breakpoints in methods disable native code. Also, a session with a very large `GEM_TEMPOBJ_CACHE_SIZE`, on the Mac, may disable native code for internal reasons.

The runtime parameter `#GemNativeCodeEnabled` can be used to disable native code, and can be used to control whether subsequent `GsProcesses` start execution using interpreted or native code. Enabling generation of native code for methods as they are loaded for execution can only be controlled by the value of `GEM_NATIVE_CODE_ENABLED` in the configuration files at process startup.

- ▶ 0 or `FALSE` disables native code generation
- ▶ 1 enables native code generation
- ▶ 2 or `TRUE` enables native code generation, with inlining of some `SmallInteger` math primitives.

Runtime parameter: `#GemNativeCodeEnabled`

Minimum: 0

Maximum: 2

Default: 2

GEM_PGSRV_COMPRESS_PAGE_TRANSFERS

If `GEM_PGSRV_COMPRESS_PAGE_TRANSFERS` is true, use `LZ4_compress()` from the LZ4 compression library to compress page transfers between the page server on Stone's machine and Gem or mid-cache page server.

For the first Gem to login on a remote machine, that Gem's configuration file value of `GEM_PGSRV_COMPRESS_PAGE_TRANSFERS` is propagated to the page manager, and is used to configure the page manager's communication to the page manager's pgsrv on the new remote cache.

When a Gem triggers creation of a mid-level cache via the method `midLevelCacheConnect:cacheSizeKB:maxSessions:`, that Gem's current runtime value of `GEM_PGSRV_COMPRESS_PAGE_TRANSFERS` is propagated to the page manager, and is used to configure the page manager's communication to the page manager's pgsrv on the new mid-level cache.

Runtime parameter: `#GemPgsrvCompressPageTransfers`
Default: FALSE

GEM_PGSRV_FREE_FRAME_CACHE_SIZE

`GEM_PGSRV_FREE_FRAME_CACHE_SIZE` specifies the size of the free frame cache used by the Gem's remote page server. This configuration option has no effect for Gems that are local to the repository extents (which have a page server).

When using the free frame cache, the page server removes enough frames from the free frame list to refill the cache in a single operation. When adding frames to the free list, the page server does not add them until the cache is full.

A value of 0 disables the free frame cache (the page server acquires frames one at a time). A value of -1 means use the default value: 0 for caches less than 100 MB and 10 for caches of 100 MB or greater.

Cache Statistic: `FreeFrameCacheSize` (Page Server)
Units: frames
Min: -1
Max: 63
Default: -1 (see above discussion)

GEM_PGSRV_FREE_FRAME_LIMIT

`GEM_PGSRV_FREE_FRAME_LIMIT` determines the free frame limit used by the Gem's remote page server. It has no effect for Gems local to the repository extents (which do not have a page server). For a description of free frames, see the configuration option `GEM_FREE_FRAME_LIMIT` (page 297).

If the value of `GEM_PGSRV_FREE_FRAME_LIMIT` is -1, the free frame limit is set to one of the following default values:

- ▶ For primary shared page cache that is 800 MB or smaller: 10% of the number of frames in the cache
- ▶ For primary shared page cache greater than 800 MB: 5000 frames

To tune the free frame limit of a page server at runtime, use the method `System class>>changeCacheSlotFreeFrameLimit: aSlot to: aValue`.

Cache Statistic: **FreeFrameLimit** (Page Server)
Min: -1
Max: 65536
Default: -1 (see above discussion)

GEM_PGSRV_UPDATE_CACHE_ON_READ

GEM_PGSRV_UPDATE_CACHE_ON_READ determines the read behavior of the Gem's remote page server when pages are read from disk. If this option is set to true, pages read from disk are also added to the shared page cache on the page server's host. If this option is false, pages read are not added to the page server's shared cache.

This option has no effect for Gems that are local to the repository extents, which do not have page servers, nor on mid-level caches.

Runtime parameter: **#GemPgsvrUpdateCacheOnRead**
Default: false

GEM_PGSRV_USE_SSL

GEM_PGSRV_USE_SSL controls whether a remote gem uses a secure socket layer (SSL) connection to converse with its page server(s), both the page server on the stone's host and the page server on the mid-level cache, if any.

This option has no effect local gems (i.e., gems running on the same host the stone process).

Secure sockets are slightly slower than insecure sockets due to the overhead of encrypting and decrypting data.

When enabling this option, consider also enabling the GEM_PGSRV_COMPRESS_PAGE_TRANSFERS option. SSL encodes no more than 16 KB into a single packet and some messages sent between the gem and page server may exceed this limit, causing SSL to send multiple encrypted packets to convey a single message.

Default: FALSE

GEM_PRIVATE_PAGE_CACHE_KB

GEM_PRIVATE_PAGE_CACHE_KB sets the size of the Gem's private page cache. This setting also applies to linked Gems.

If no units are specified, the value is in KB. You may also specify units as KB, MB, or GB.

Default units: KB
Min: 128
Max: 524288
Default: 1000

GEM_READ_AUTH_ERR_STUBS

GEM_READ_AUTH_ERR_STUBS configures the behavior when a read authorization denied occurs on object fault. When FALSE (the default), a SecurityError is signalled. When set to TRUE, an in-memory instance of UnauthorizedObjectStub is constructed.

Runtime equivalent: **#GemReadAuthErrStubs**
Default: FALSE

GEM_REPOSITORY_IN_MEMORY

GEM_REPOSITORY_IN_MEMORY sets the performance behavior of the gem for certain operations that scan the entire repository. If set to TRUE, the gem assumes most or all of the data pages in the repository have been previously loaded into the shared page cache. If set to FALSE, the gem assumes most or all of the data pages in the repository are not in the shared page cache and must be read from disk.

This setting affects performance only. All operations affected by this setting will succeed and produce the same results.

Repository instance methods affected by this setting include:

`findReferencePathToObject`: (and related methods)

`findAllReferencePathsToObjects`: (and related methods))

Runtime parameter: `#GemRepositoryInMemory`

Default: FALSE

GEM_RPCGCI_TIMEOUT

GEM_RPCGCI_TIMEOUT specifies the time in minutes after which lack of an Rpc command will cause a Gem to terminate. Negative timeouts are not allowed. Resolution of timeouts is one-half the specified timeout interval.

Min: 0

Default: 0 (Gem waits forever)

GEM_RPC_KEEPALIVE_INTERVAL

GEM_RPC_KEEPALIVE_INTERVAL is the interval in seconds for the RPC GCI client to send a packet to the gem to ensure the network connection is kept alive.

With the traditional GCI interface (`gci.hf`), this controls how often keep-alive packets are sent during the `GciPollForSignal()` calls. The application needs to be calling `GciPollForSignal` at regular intervals at least as often as the configured value.

In the thread safe GCI (`gcits.hf`), the gem will send an interrupt byte periodically, and the application must be calling `GciTsWaitForEvent` at least as often as this config value.

Min: 0

Max: 7200

Default: 0 (disabled)

GEM_RPC_USE_SSL

GEM_RPC_USE_SSL controls whether a remote RPC gem uses a secure socket layer (SSL) connection to converse with its RPC client. RPC sessions always establish a secure connection during the login sequence. This parameter controls whether the gem and its remote RPC client continue using the SSL connection. Otherwise, a standard TCP/IP socket connection is used.

This option has no effect for linked gems and local RPC gems (i.e., a gem running on the same host its client). Local gems always revert to a standard TCP/IP socket after login.

Secure sockets are slightly slower than insecure sockets due to the overhead of encrypting and decrypting data.

Default: TRUE

GEM_STATMONITOR_ARGS

GEM_STATMONITOR_ARGS describes a list of arguments to be passed to the statmonitor executable. If the list is not an empty string, statmonitor will be started automatically by a remote gem which creates a remote shared page cache; this setting is ignored other than by remote gems that create a remote shared page cache.

Example: start statmonitor with a 5 second sample interval with a date and timestamp included in the output file name and place the output file in the directory

/statmonFiles:

```
GEM_STATMONITOR_ARGS = "-i5 -u5 -d /statmonFiles
-F'statmon_%%S_%%P_%d-%m-%y-%H:%M:%S'";
```

GEM_STATMONITOR_MID_CACHE_ARGS

GEM_STATMONITOR_MID_CACHE_ARGS describes a list of arguments to be passed to the statmonitor executable. If the list is not an empty string, statmonitor will be started automatically by the remote gem which creates a mid-level shared page cache; this setting is ignored other than by remote gems that create a mid-level shared page cache.

Example: start statmonitor with a 5 second sample interval with a date and timestamp included in the output file name and place the output file in the directory

/statmonFiles:

```
GEM_STATMONITOR_MID_CACHE_ARGS = "-i5 -u5 -d /statmonFiles
-F'statmon_%%S_%%P_%d-%m-%y-%H:%M:%S'";
```

GEM_SOFTREF_CLEANUP_PERCENT_MEM

GEM_SOFTREF_CLEANUP_PERCENT_MEM controls the cleanup of SoftReferences.

If *startingMemUsed* – the percentage of temporary object memory in-use at the beginning of a VM mark/sweep – is less than the value of this option, no SoftReferences will be cleared.

If *startingMemUsed* is greater than the value of this option and less than 80%, the VM mark/sweep will attempt to clear an internally determined number of least recently used SoftReferences. Under rare circumstances, you might choose to specify a minimum number (GEM_KEEP_MIN_SOFTREFS) that will not be cleared.

If *startingMemUsed* is greater than 80%, VM mark/sweep will attempt to clear all SoftReferences.

Also see the statistics **NumSoftRefsCleared**, **NumLiveSoftRefs**, and **NumNonNilSoftRefs**.

Runtime parameter: **#GemSoftRefCleanupPercentMem**

Min: 10

Max: 80

Default: 50

GEM_TEMPOBJ_AGGRESSIVE_STUBBING

GEM_TEMPOBJ_AGGRESSIVE_STUBBING controls stubbing in in-memory garbage collection. If instance variable X in object A references object B, and X contains a memory pointer to B, then the reference is *stubbed* by storing into instance variable X the objectId of object B.

When this option is TRUE (the default), references from temporary objects to in-memory copies of committed objects are stubbed whenever possible, during both scavenge and mark/sweep. Also, references from not-dirty in-memory copies of committed objects to other committed objects are stubbed whenever possible. This reduces the number of committed objects forced to stay in-memory, but can slow down garbage collection and subsequent execution.

When this option is FALSE, references from temporary objects to in-memory copies of committed objects are never stubbed. References from not-dirty in-memory copies of committed objects to other committed objects are stubbed after the number of objects flushed during commits reaches a threshold, or if almost OutOfMemory. Performance may be faster, but there is a greater risk of OutOfMemory errors.

Stubbing is always disabled when a commit attempt is in progress, regardless of the setting of this parameter. Certain objects private to the object manager are always immune from stubbing, and so are references stored into Session State by using `System class >> _sessionStateAt:put:.`

Also see the statistics **NumRefsStubbedMarkSweep** and **NumRefsStubbedScavenge**.

Default: true

GEM_TEMPOBJ_CACHE_SIZE

GEM_TEMPOBJ_CACHE_SIZE sets the maximum size of the Gem's temporary object memory. This limit also applies to memory in linked Topaz sessions and linked GemBuilder applications. This value is set when the VM is initialized and cannot be changed without restarting the VM. When you only change this setting, and the other GEM_TEMPOBJ* configuration options use default values, then all of the various spaces remain in proportion to each other.

This setting defines the maximum memory size. The initial memory allocated will be smaller, and as the actual space required for objects grows, the VM requests and allocates virtual memory as needed. As the limit is approached, in-memory garbage collection becomes more aggressive; if the limit is reached, the Gem will exit.

If no units are specified, the value is in KB. You may also specify units as KB, MB, or GB.

Statistic: **GemTempObjCacheSizeKb** (Gem)

Default Units: KB

Min: 2000 KB

Max: 62 GB

Default: 50000 KB

GEM_TEMPOBJ_CONSECUTIVE_MARKSWEEP_LIMIT

GEM_TEMPOBJ_CONSECUTIVE_MARKSWEEP_LIMIT controls in-memory garbage collection. If there are more consecutive in-memory mark sweeps than this value, without any intervening successful in-memory scavenges, then an OutOfMemory error will occur.

Runtime equivalent: **GemTempObjConsecutiveMarkSweepLimit** (Gem)

Default: 50

Min: 20

Max: 5000

GEM_TEMPOBJ_MESPACE_SIZE

GEM_TEMPOBJ_MESPACE_SIZE sets the maximum size of the Map Entries space within the Gem's temporary object memory. This value is set when the VM is initialized and cannot be changed without starting the VM.

One Map Entry is required for each faulted-in committed object, or for any temporary object that might become committed, referenced from an IdentityBag, or exported to the GCI. One Map Entry occupies approximately 24 bytes.

If a Map Entry is needed and the Map Entries Space is full, an OutOfMemory occurs, terminating the session.

Unless you are trying to minimize the memory footprint on HP-UX or AIX, you should always leave GEM_TEMPOBJ_MESPACE_SIZE at its default value (0) so that the system can calculate the size of the Map Entries space based on other memory sizes. Otherwise, you are at risk of premature OutOfMemory errors.

If no units are specified, the value is in KB. You may also specify units as KB, MB, or GB.

Default units: KB
Min: 1000 KB
Max: 1000000 KB
Default: 0

GEM_TEMPOBJ_OOMSTATS_CSV

GEM_TEMPOBJ_OOMSTATS_CSV configures the production of a CSV file on an OutOfMemory error. If TRUE, then when an Error 4067/OutOfMemory occurs, the detailed statistics of instances of classes in temporary memory are written to a `gemnetobjectpid.csv` file in CSV format. For a `topaz -l` process, CSV data is written to a `topazpid.csv` file in the current directory.

The config parameter GEM_TEMPOBJ_OOMSTATS_CSV and the environment variable GS_DEBUG_VMGC_VERBOSE_OUTOFMEM operate independently. The environment variable GS_DEBUG_VMGC_VERBOSE_OUTOFMEM controls writing statistics in standard text to the gem log or stdout.

Runtime parameter: `#GemTempObjOomstatsCsv`
Default: FALSE

GEM_TEMPOBJ_OOPMAP_SIZE

GEM_TEMPOBJ_OOPMAP_SIZE sets the size of the hash table (that is, the number of 8-byte entries) in the objId-to-object map within the Gem's temporary object memory. This value is set when the VM is initialized and cannot be changed without starting the VM.

The value specified is rounded up to the next higher power of 2.

This option should normally be left at its default value (0) so that the system can calculate the size of the map based on other memory sizes.

Min: 16384
Max: 524288000
Default: 0

GEM_TEMPOBJ_POMGEN_PRUNE_ON_VOTE

GEM_TEMPOBJ_POMGEN_PRUNE_ON_VOTE sets the percent of POM generation area to be thrown away when voting on possible dead objects.

If the value is 0, no subspaces of POM generation are cleared; if the value is 100, all subspaces are cleared. For values greater than 0 and less than 100, the number of spaces that are in use and older than 5 minutes is computed, and the specified parameter is the percentage, rounded down, of these subspaces that are discarded.

Runtime parameter: **#GemPomGenPruneOnVote**

Default: 50

Min: 0

Max: 100

GEM_TEMPOBJ_POMGEN_SCAVENGE_INTERVAL

GEM_TEMPOBJ_POMGEN_SCAVENGE_INTERVAL is the interval in seconds in which the oldest POM generation subspace will be discarded. Lower values may reduce Gem memory usage but may also cause objects to be re-read. Larger values may result in higher Gem memory usage and may reduce disk I/O. Setting this value to zero disables scheduled POM generation scavenges. In this case, POM generation will only be scavenged when all subspaces become full.

Runtime parameter: **#GemTempObjPomgenScavengeInterval**

Units: seconds

Min: 0

Max: 86400

Default: 1800

GEM_TEMPOBJ_POMGEN_SIZE

GEM_TEMPOBJ_POMGEN_SIZE sets the maximum size of the POM generation area in the Gem's temporary object memory. This value is set when the VM is initialized and cannot be changed without starting the VM.

The POM generation area holds unmodified copies of committed objects that have been faulted into a Gem, and is divided into ten subspaces.

This option should normally be left at its default value (0) so that the POM generation area is allocated to the default, which is approximately 0.8 times the size of the GEM_TEMPOBJ_CACHE_SIZE.

If no units are specified, the value is in KB. You may also specify units as KB, MB, or GB.

Default units: KB

Min: 1000 KB

Max: 1000000 KB

Default: 0

GEM_TEMPOBJ_SCOPES_SIZE

GEM_TEMPOBJ_SCOPES_SIZE is the size of the scopes stack in Gem temporary object garbage collection. This value is set when the VM is initialized and cannot be changed without starting the VM.

The scopes stack consumes (8 bytes * GEM_TEMPOBJ_SCOPES_SIZE) of C heap memory.

The primary user-visible effect of this setting is maximum depth of nested expressions that can be compiled by the method compiler. The default setting is sufficient for expression nesting of about 200, such as in depth of nested parenthesized expressions.

Min: 1000

Max: 10000000

Default: 2000

GEM_TEMPOBJ_START_ADDR

GEM_TEMPOBJ_START_ADDR applies for AIX only. If the default mmap of temporary object memory fails, this value is used to define the starting address at which to attempt to mmap temporary object memory using MAP_FIXED at fixed addresses and munmap to simulate MAP_NORESERVE.

A config file value of zero results in an internal default of 0xA000000000000000 for AIX 7, and 0x7000000000000000 for AIX 6. A non-default value must be coded as an exact address and may be affected by use of mmap by other shared libraries.

Default: 0

INCLUDE

INCLUDE specifies the name of a file to be included in this config file when parsing include files. The value must be a single string, which is a path to a file, and may include environment variables which will be expanded. INCLUDE directives may be nested up to 100 deep.

Unlike other configuration parameters, the INCLUDE parameter may be present more than once in a configuration file, and each file specified by an INCLUDE, not just the last one, is read. When other configuration parameters settings are found in more than one file, however, the last one read is the value that is used.

For example:

```
INCLUDE=config2.conf;    # will look in current directory
INCLUDE="$GEMSTONE/data/config3.conf";
```

KEYFILE

KEYFILE sets the location of GemStone licensing keyfile.

Default: \$GEMSTONE/sys/gemstone.key

LOG_WARNINGS

If LOG_WARNINGS is set to true, warnings are printed for invalid configuration options.

Default: true

SHR_NUM_FREE_FRAME_SERVERS

SHR_NUM_FREE_FRAME_SERVERS specifies the number of free frame page server threads that will be started when the shared page cache is created. A value of -1 means the default value should be used. On the primary shared page cache (the cache to which the stone attaches), the default value is equal to the value used for the STN_NUM_LOCAL_AIO_SERVERS parameter. On a remote shared page cache, the default is 1.

Min: -1
Max: 255
Default: -1

SHR_PAGE_CACHE_LARGE_MEMORY_PAGE_POLICY

SHR_PAGE_CACHE_LARGE_MEMORY_PAGE_POLICY specifies whether large memory pages will be used when creating the shared page cache. Enabling large memory pages can result in significant performance gains when large shared page caches are used. The improvement is due to a reduction in translation lookaside buffer (TLB) cache misses. The TLB is an internal structure used by the operating system to manage memory address translation.

Large memory page support is an operating system and hardware dependent feature. Currently, GemStone supports large memory pages on AIX and Linux only. This configuration option is silently ignored on all other platforms.

Three policies are available on supported operating systems:

- 0 - Disabled: No large memory page support.
- 1 - Advisory: Large memory pages are requested when the shared page cache is created. If the operating system denies the request, a warning is printed in the SPC monitor log file and the cache is started without large memory pages. **It is strongly discouraged to use this setting on AIX.**
- 2 - Mandatory: Large memory pages are requested when the shared page cache is created. If the operating system denies the request, an error is printed in the SPC monitor log file and the shared page cache fails to start.

Both Linux and AIX require operating system kernel changes in order to enable large memory pages. Refer to the *Installation Guide* for your platform for more information.

Default: 0
Minimum: 0
Maximum: 2

SHR_PAGE_CACHE_LARGE_MEMORY_PAGE_SIZE_MB

On Linux only, specifies the large memory page size, in megabytes, that will be used when creating the shared page cache. A value of 0 means use the default large page size for the host. This option is only supported on Linux, and any setting is ignored on other platforms.

This setting is ignored if SHR_PAGE_CACHE_LARGE_MEMORY_PAGE_POLICY is set to 0.

Valid large page sizes on Linux are 2 MB and 1024 MB; not all systems will support both large memory page sizes. Normally the default large page size on Linux is 2 MB, but that default can be changed by the system administrator.

Default: 2
Minimum: 0
Maximum: 1024

SHR_PAGE_CACHE_LOCKED

SHR_PAGE_CACHE_LOCKED specifies whether the shared page cache should be locked in memory. On systems that permit a portion of memory to be dedicated to GemStone, this option may provide higher performance.

On Solaris 10, GemStone uses Intimate Shared Memory for the shared page cache, making setting this variable unnecessary.

Other specific operating systems may restrict this action to processes running as root or may require special privileges (such as on Linux, either the capability CAP_IPC_LOCK or an RLIMIT_MEMLOCK resource limit greater than the size of the shared page cache). For further information, check the shared page cache monitor log for error messages and consult your operating system documentation.

Default: false

SHR_PAGE_CACHE_NUM_PROCS

SHR_PAGE_CACHE_NUM_PROCS sets the maximum number of processes allowed to attach to the shared page cache. This parameter is used to allocate space in the shared page cache for session information and cache statistics. This cache space is in addition to extent page space allocated by SHR_PAGE_CACHE_SIZE_KB.

The value for SHR_PAGE_CACHE_NUM_PROCS must accommodate the GcGems and various background GemStone processes, as well as user Gem and Topaz session processes. If the value is too small, sessions might be unable to login because they can't attach to the cache. If the value is too large, space in the cache may be wasted.

It is recommended to leave this at the default value. When the default setting of -1 is specified, the value of this parameter is calculated as:

```
STN_MAX_SESSIONS
+ 8 (for system logins)
+ STN_MAX_GC_RECLAIM_SESSIONS
+ SHR_NUM_FREE_FRAME_SERVERS
+ STN_NUM_LOCAL_AIO_SERVERS

Cache Statistic: (SPC Monitor) SlotsTotalCount
Min: 15, or the number extents + 3, whichever is larger
Max: determined by STN_MAX_SESSIONS or file descriptor limits
Default: -1
```

SHR_PAGE_CACHE_NUM_SHARED_COUNTERS

SHR_PAGE_CACHE_NUM_SHARED_COUNTERS specifies the number of shared counters available in the shared page cache. On most platforms, each counter consumes 128 bytes of shared memory. On AIX, each counter consumes 256 bytes of shared memory. Shared memory used for shared counters is in addition to the shared memory size specified in SHR_PAGE_CACHE_SIZE_KB.

```
Cache Statistic: NumSharedCounters (SPC Monitor)
Min: 0
Max: 500000
Default: 1900
```

SHR_PAGE_CACHE_PERMISSIONS

SHR_PAGE_CACHE_PERMISSIONS specifies the UNIX permission settings of the shared page cache, expressed as an octal number. The first two digits are constant and must always be 06. The 0 indicates an octal constant and 6 indicates the UNIX user which created the cache has read/write permissions.

The last two digits specifies the group and other permissions respectively. Each of the last two digits must be one of the following:

- 6 - read/write
- 4 - read only
- 0 - no access

By default, the shared page cache is created with group read/write permission but no access for other users.

Default: 0660

SHR_PAGE_CACHE_SIZE_KB

SHR_PAGE_CACHE_SIZE_KB sets the base size of the shared page cache. Additional shared memory is used for overhead, so the actual size of the memory segment will be somewhat larger.

If no units are specified, the value is in KB. You may also specify units as KB, MB, or GB.

Default Units: KB

Min: 16000 KB

Max: Limited by system memory and kernel configurations

Default: 75000 KB

NOTE

For information about platform-specific limitations on the size of the shared page cache, refer to Chapter 1 of your GemStone/S Installation Guide.

SHR_SPIN_LOCK_COUNT

SHR_SPIN_LOCK_COUNT specifies the number of tries to get a spin lock before the process sleeps on a semaphore. Semaphores involve a relatively time-consuming call to the operating system. Spin locks involve busy-wait loops. Efficient locking may require a combination of these methods.

In single-processor architectures, this value should always be 1 since there is no value in spinning (the lock won't change until the process holding the lock gets scheduled). On multiple-processor architectures, a value of 5000 is recommended.

We recommend that you leave this option set to the default value of -1, which causes GemStone to use a value of either 1 or 5000, based upon the number of CPUs detected.

The runtime parameter can be changed only by SystemUser.

Runtime parameter: **#SpinLockCount**

Default: -1 (use either 1 or 5000, based on the number of CPUs detected)

SHR_TARGET_FREE_FRAME_COUNT

SHR_TARGET_FREE_FRAME_COUNT specifies the target number of free frames to keep in the shared cache at all times. The free frame page server process(es) will attempt to keep the number of free frames in the cache equal to or greater than this value.

If the value of the parameter is -1, the target free frame count is set to a percentage of the total frames in the shared cache. For the main shared cache (the cache to which the stone attaches), the default is 1/8 the frames in the cache. For remote caches, the default is 1/100 the frames in the cache.

For best performance, keep this setting greater than GEM_FREE_FRAME_LIMIT.

If the value of SHR_TARGET_FREE_FRAME_COUNT is -1, the target number of free frames is set to one of the following default values:

- ▶ For primary shared page cache that is 800 MB or smaller: 12.5% of the number of frames in the cache
- ▶ For primary shared page cache greater than 800 MB: 7000 frames
- ▶ For a remote shared page cache: 1% of the number of frames, or 2000 frames, whichever is smaller

Min: -1

Max: 65536

Default: -1 (see above discussion)

SHR_WELL_KNOWN_PORT_NUMBER

SHR_WELL_KNOWN_PORT_NUMBER specifies the port number that the shared page cache monitor will use as its well-known port. The well-known port is used by all Gems and page servers on this host to connect to the cache monitor.

If the specified port is in use by another process, the monitor process will not start and exits with an error. A value of zero indicates that the port number will be selected by the system.

Min: 1

Max: 65535

Default: 0

STN_ADMIN_GC_SESSION_ENABLED

STN_ADMIN_GC_SESSION_ENABLED determines whether the Admin Gem is started when the Stone is started. (The Admin Gem performs administrative garbage collection functions such as write set union sweeps; the Reclaim Gem performs dead object and page reclaim.)

Runtime parameter: **#StnAdminGcSessionEnabled**

Default: true

STN_ALLOCATE_HIGH_OOPS

STN_ALLOCATE_HIGH_OOPS instructs the Stone to skip the first 16 million object identifiers and begins to allocate object identifiers (GCI OopTypes) for non-special objects at 16r100000001.

This option is designed for testing conversion of GCI applications and user actions. Do not set this option in a production environment.

Default: FALSE

STN_ALLOW_NFS_EXTENTS

STN_ALLOW_NFS_EXTENTS allows the Stone to start up using extents and tranlogs which are on NFS-mounted filesystems. This is less reliable and less performant than locally mounted filesystems, or filesystems on storage arrays which appear as local mounts. This variable cannot be changed at runtime.

Default: FALSE

STN_ALLOW_NO_SESSION_INIT

STN_ALLOW_NO_SESSION_INIT enables bypass, and allows debugging, of `GsCurrentSession>>initialize` and any functions specified by `UserProfile's loginHook`, which are normally always executed on login.

If false, `GciLogin` ignores the flag `GCI_CLIENT_DOES_SESSION_INIT`. All sessions perform `GsCurrentSession>>initialize`, and errors are fatal to the session logging in.

If true, `GciLogin` uses the setting for the flag `GCI_CLIENT_DOES_SESSION_INIT`. `GsCurrentSession>>initialize` is executed unless this flag is set or `topaz` executes the `SET SESSIONINIT OFF` command. If `GsCurrentSession>>initialize` is executed, errors in execution are reported, but are not fatal errors.

See also the `topaz` command `set sessioninit off` in the *Topaz Programming Guide*.

Runtime parameter: `#StnAllowNoSessionInit` (may only be set by `SystemUser`)

Default: FALSE

STN_CACHE_WARMER

Legacy; replaced by `STN_CACHE_WARMER_ARGS`. Specifies the type of cache warming to perform on startup.

The `STN_CACHE_WARMER` has the following possible values:

- 0 - Disabled; with `STN_CACHE_WARMER_ARGS`, specify ""
- 1 - Warm only object table pages; with `STN_CACHE_WARMER_ARGS` to " "
- 2 - Warm object table and data pages; with `STN_CACHE_WARMER_ARGS`, include the flag "-d".

STN_CACHE_WARMER_ARGS

This configuration option can be used to have the stone automatically start a cache warmer process on its cache. The options that can be specified here are:

- d read data pages into the cache.
- D read data pages into the UNIX file system buffers
- l stop cache warming if the free frame count falls below `<cacheFullLimit>`.
- n the number of warmer threads to use
- w enable saving and restoring a `workingSet` of data pages

The complete description of the options and default values can be obtained by invoking the command: `$GEMSTONE/bin/startcachewarmer -h`

The other options if specified are ignored by the cache warmer when starting from this configuration. The `-s stoneName` argument is automatically provided, so it is not required.

If this configuration option is specified with a non empty string, it overrides any settings for `STN_CACHE_WARMER` and `STN_CACHE_WARMER_SESSIONS`

Example: Start a cachewarmer to load only the object table into the cache using 4 warmer threads.

```
STN_CACHE_WARMER_ARGS = "-n 4"
```

Default: "" (cachewarmer is not started)

STN_CACHE_WARMER_SESSIONS

Legacy; replaced by `STN_CACHE_WARMER_ARGS`. Specifies the number of worker sessions (threads) to use to perform cache warming on startup. A value of 0 means to compute the default based on (numberOfCPUs + numberOfExtents).

Equivalent to using `-n numSessions` in the `STN_CACHE_WARMER_ARGS`.

STN_CACHE_WARMER_WAIT_MODE

Determines if and when startstone waits until cache warming has finished. The following values are allowed:

- 0 - disabled. startstone does not wait until cache warming has finished.
- 1 - startstone waits until cache warming has finished but only if stone is starting after a clean shutdown, or without tranlogs (i.e., startstone with `-N` option). Otherwise startstone does not wait for cache warming to finish. This is the default setting.
- 2 - startstone always waits until cache warming has finished.

Has no effect and is ignored if cache warming is not enabled.

Minimum: 0

Maximum: 2

Default: 1

STN_CHECKPOINT_INTERVAL

`STN_CHECKPOINT_INTERVAL` sets the maximum interval between checkpoints. Checkpoints may be written more often, depending on other factors. The unit is seconds.

This can be changed at runtime only by SystemUser.

Runtime parameter: **#StnCheckpointInterval**

Units: seconds

Min: 5

Max: 1800

Default: 300

STN_COMMIT_QUEUE_THRESHOLD

`STN_COMMIT_QUEUE_THRESHOLD` determines whether the Stone defers the disposal of commit records, based on the number of sessions in the commit queue and the run queue. If the size of either of these queues exceeds this threshold, the Stone defers commit record disposal until all queues have sizes less than or equal to the value.

This setting is ignored if the commit record backlog exceeds the value of `STN_CR_BACKLOG_THRESHOLD`.

Runtime parameter: **#StnCommitQueueThreshold**

Default: -1 (never defer commit record disposal)

Min: -1
Max: 1024

STN_COMMIT_RECORD_BM_CACHING

When true, enables caching at the commit point of page allocation information needed when disposing a commit record. This option can reduce I/Os during commit record dispose when the commit record backlog is high, or when there is a lot of page preemption occurring in the shared cache.

When enabled, the maximum commit rate is slightly lower and commit latency is slightly higher because more work is done in the commit critical region in stone.

Default: true

STN_COMMIT_RECORD_QUEUE_SIZE

STN_COMMIT_RECORD_QUEUE_SIZE determines the size of the Stone's internal commit record cache. The Stone will keep copies of up to this many commit records in heap memory. Stone is able to dispose commit records more quickly when a copy of the commit record is found in this cache.

When the default value of -1 is specified, Stone sets this value to be twice the value of the STN_SIGNAL_ABORT_CR_BACKLOG option.

Units: Commit Record Pages
Default: -1
Min: 16
Max: 1000000

STN_COMMIT_TOKEN_TIMEOUT

STN_COMMIT_TOKEN_TIMEOUT sets the maximum interval (in seconds) that a session may possess the commit token. If the session possesses the token for longer than this period, the session will be logged off the system and an error message written to the Stone log. If the value is non zero, GcGems of all types will have a timeout of twice the configured value.

Default: 0 (Stone waits forever)
Min: 0
Max: 86400

STN_COMMITS_ASYNC

If STN_COMMITS_ASYNC is set to TRUE, it causes the stone to acknowledge each commit or persistent shared counter update to the requesting session without waiting for the tranlog writes for that commit to complete.

Default: FALSE

STN_CR_BACKLOG_THRESHOLD

STN_CR_BACKLOG_THRESHOLD sets the size of the commit record backlog above which the Stone aggressively disposes of commit records. This setting overrides the deferral of commit record disposal provided by the STN_COMMIT_QUEUE_THRESHOLD parameter.

The default setting of -1 causes the Stone to use a setting equal to (2 * STN_MAX_SESSIONS). A setting of 0 disables this threshold.

Runtime parameter: **#StnCrBacklogThreshold**
Default: -1
Min: -1
Max: 500000

STN_DISABLE_LOGIN_FAILURE_LIMIT

STN_DISABLE_LOGIN_FAILURE_LIMIT is the number of failed login attempts, within the time limit set by STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT, that result in the user account being disabled. When an account exceeds these limits, the user account is disabled (the system changes the password on the account to one that is invalid) and a record of the event is written to the Stone log file. The user account can only be restored by another user with OtherPassword privileges.

Changes to the runtime parameter requires the OtherPassword privilege.

Runtime parameter: **#StnDisableLoginFailureLimit**
Units: login attempts
Default: 15
Min: 0
Max: 65536

STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT

STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT is the period of time in which if STN_DISABLE_LOGIN_FAILURE_LIMIT number of failed login attempts are made, the user account will be disabled (the system changes the password on the account to one that is invalid) and a record of the event is written to the Stone log file. The user account can only be restored by another user with OtherPassword privileges.

Changes to the runtime parameter requires the OtherPassword privilege.

Runtime parameter: **#StnDisableLoginFailureTimeLimit**
Units: Minutes
Default: 15
Min: 1
Max: 1440 (24 hours)

STN_DISKFULL_TERMINATION_INTERVAL

STN_DISKFULL_TERMINATION_INTERVAL specifies how soon (in minutes) the Stone should start terminating sessions holding on to the oldest commit record when the repository free space is below the value set for STN_FREE_SPACE_THRESHOLD. Such sessions are sent the fatal diskfull error.

The runtime parameter can be changed only by SystemUser.

Runtime parameter: **#StnDiskFullTerminationInterval**
Units: Minutes
Min: 0 (no sessions are terminated)
Max: 1440 (24 hours)
Default: 3

STN_EPOCH_GC_ENABLED

STN_EPOCH_GC_ENABLED determines if epoch garbage collection can be run on the system. Leave this value set to FALSE unless you plan to run epoch garbage collection on the system. Setting this to TRUE adds a small amount of overhead to commit processing.

Runtime parameter: **#StnEpochGcEnabled**

Default: FALSE

STN_EXTENT_IO_FLAGS

STN_EXTENT_IO_FLAGS specifies what (if any) special I/O flags will be used to open the database extents. Two kinds of special I/O are supported: direct I/O and concurrent I/O.

Direct I/O tells the operating system avoid caching extent data in the file system cache. Enabling direct I/O tells the operating system to treat the database extents as if they were on raw partitions. Direct I/O may greatly improve database performance in some cases. Concurrent I/O is only available to extents running on the Enhanced JFS file system (aka JFS2) on AIX. Setting this flag has no effect on other operating systems.

STN_EXTENT_IO_FLAGS has the following possible values:

0 - no special I/O flags. This is the default.

1 - enable Direct I/O on all extents on file systems.

2 - enable concurrent I/O (AIX only)

If the requested I/O mode is not available, the stone will fail to start and an error message will be printed in the stone log. If this happens, change this option back to zero and restart the stone.

STN_EXTENT_IO_FLAGS has no effect on extents which reside on raw partitions.

Once the stone starts, all processes which open the database extents (gems and page servers) will open the extents using the same I/O flags. This behavior is required by some operating systems.

Default: 0

Min: 0

Max: 2 (AIX only), 1 (All Others)

STN_FREE_FRAME_CACHE_SIZE

STN_FREE_FRAME_CACHE_SIZE specifies the size of the Stone's free frame cache. When using the free frame cache, the Stone removes enough frames from the free frame list to refill the cache in a single operation.

Units: frames

Default: 1 (disables the free frame cache; Stone acquires frames one at a time)

Min: 1

Max: 1% of the frames in the cache

STN_FREE_SPACE_THRESHOLD

STN_FREE_SPACE_THRESHOLD sets the minimum amount of free space to be available in the repository. If the Stone cannot maintain this level by growing an extent, it begins to take action to prevent the shutdown of the system. If the amount of free space remains below this level for more than the number of minutes specified by

STN_DISKFULL_TERMINATION_INTERVAL, the stone will start terminating sessions. For more information, see “Repository Full” on page 196.

The default value of 0 specifies a varying STN_FREE_SPACE_THRESHOLD that is computed when needed as the current size of the repository divided by 1000, with a minimum value of 5 MB.

If no units are specified, the value is in MB. You may also specify units as KB, MB, or GB.

The runtime parameter can be changed only by SystemUser.

Runtime parameter: **#StnFreeSpaceThreshold**
Default units: MB
Default: 0 (system computes based on repository size)
Min: 0
Max: 65536 MB

STN_GEM_ABORT_TIMEOUT

STN_GEM_ABORT_TIMEOUT sets the time in minutes that the Stone will wait for a Gem running outside of a transaction to abort (in order to release a commit record), after Stone has signaled that Gem to do so. If the time expires before the Gem aborts, the Stone sends the Gem the error ABORT_ERR_LOST_OT_ROOT, and then either stop the Gem or force it to completely reinitialize its object caches, depending on the value of the related configuration parameter STN_GEM_LOSTOT_TIMEOUT. Negative timeouts are not allowed. Resolution of timeouts is one half the specified time-out interval.

The runtime parameter can be changed only by SystemUser.

Runtime parameter: **#StnGemAbortTimeout**
Default: 1
Min: 1
Max: 1440

STN_GEM_LOSTOT_TIMEOUT

STN_GEM_LOSTOT_TIMEOUT sets the time in seconds that the Stone will wait after signaling the Exception `RepositoryViewLost`, before retracting the Gem’s commit record and forcibly stopping the session.

If set to zero, no signal is sent; the stone immediately stops the session.

The runtime parameter can be changed only by SystemUser.

Runtime parameter: **#StnGemLostOfTimeout**
Default: 60
Min: 0
Max: 5000000

STN_GEM_PGSRV_CONNECT_TIMEOUT

The time in seconds that a remote gem will wait for a connection to a pgsvr on stone's machine to complete.

Runtime equivalent: **#StnGemPgsvrConnectTimeout** (may only be set by SystemUser)
Default: 20
Min: 5
Max: 3600

STN_GEM_PRIVATE_PGSRV_ENABLED

A Boolean, TRUE means a remote gem will start a private pgsvr process if the attempt to connect to a multi-threaded pgsvr on stone's machine fails.

Runtime equivalent: `#StnGemPrivatePgsvrEnabled` (may only be set by SystemUser)
Default: FALSE

STN_GEM_TIMEOUT

STN_GEM_TIMEOUT sets the time in minutes after which lack of interaction with the Gem causes the Stone to terminate the session. Negative timeouts are not allowed. Resolution of timeouts is one half the specified time-out interval. If non-zero, this timeout is also the maximum time allowed for a Gem to complete processing of its login to the Stone. If this timeout is 0, the maximum time for login processing is set to five minutes.

The runtime parameter can be changed only by SystemUser.

Runtime parameter: `#StnGemTimeout`
Min: 0
Default: 0 (Stone waits forever)

STN_GROUP_COMMITS

A value greater than 1, specifies maximum number of commits to group into a tranlog write. Grouping is performed only if another session is waiting to commit while stone is processing a session's commit.

Default: 10
Min: 1
Max: 20

STN_HALT_ON_FATAL_ERR

If STN_HALT_ON_FATAL_ERR is set to true, the Stone will halt and dump core if it receives notification from a Gem that the Gem died with a fatal error that would cause Gem to dump core. By stopping the Stone at this point, the possibility of repository corruption is minimized. true is the recommended setting for systems during development.

If STN_HALT_ON_FATAL_ERR is set to false, the Stone will attempt to keep running if a Gem encounters a fatal error. false is the recommended setting for systems in production use.

Internally, the setting 0 = false, 1 = true. The runtime parameter can be changed only by SystemUser.

Runtime parameter: `#StnHaltOnFatalErr`
Default: false

STN_LISTENING_ADDRESSES

STN_LISTENING_ADDRESSES is a list of 0 to 10 addresses upon which stone should listen for login connections. If list is empty, the default address "::" is used, which means listen on any active network interfaces, plus the loopback ("::1") interface.

Each element of the list may be a name or a numeric IPv6 address. Each named address must resolve via `getaddrinfo()` to at least one address legal to listen on, i.e. resolve to the

loopback or wildcard address, or to an address assigned to a network interface on this machine. Each numeric address must be an address legal to listen on.

Numeric IPv6 addresses may be any form recognized by `inet_pton(AF_INET6, address, etc.)` or by `inet_pton(AF_INET, address, etc.)` on the host operating system. Per RFC 2373 this includes these forms:

- ▶ IPv4 dotted-decimal format, d.d.d.d
- ▶ IPv6 hex format x:x:x:x:x:x where x is a 16 bit hexadecimal number
- ▶ IPv4-mapped IPv6 ::FFFF:d.d.d.d where d is an 8 bit decimal number

IPv6 format may contain at most one :: which is a contiguous group of zeros. The loopback address 0:0:0:0:0:0:1 can be written as ::1. The wildcard address 0:0:0:0:0:0:0 can be written as ::.

If the list contains the wildcard address ::, the other elements of the list are ignored.

If the list does not contain ::, then the loopback addresses ::1 and 127.0.0.1 are always listened on, even if not explicitly in the list, to support logins from system gems.

See public documents RFC 4291 and RFC 4038 for more information on IPV6 addressing.

Default: An empty list, "::"

STN_LOGIN_LOG_DIR

`STN_LOGIN_LOG_DIR` specifies a directory where the login log is located when the `STN_LOGIN_LOG_ENABLED` option is set to TRUE. If `STN_LOGIN_LOG_ENABLED` is false, this option is ignored. If `STN_LOGIN_LOG_DIR` is not specified and `STN_LOGIN_LOG_ENABLED` is set to TRUE, then the login log will be placed in the same directory as the stone log. It is a fatal error if the directory specified is not writable by the stone process.

Default: Stone log's directory

Min: 0 entries

Max: 1 entry

STN_LOGIN_LOG_ENABLED

`STN_LOGIN_LOG_ENABLED` enables the logging of all session login and logout events to a separate log file owned by the stone. The file will be named `stoneName_login_timestamp.log` and will be placed in the same directory as the stone log, unless a directory is specified using `STN_LOGIN_LOG_DIR`.

When this feature is enabled, logins and logouts are recorded for all sessions by default. Logging may be disabled for a UserProfile by sending the `#disableLoginLogging` message to a UserProfile instance and committing the transaction.

The login log file is a text file that contains one line per event. Fields within a line are separated by spaces; the Timestamp String is quoted. The fields logged in each line are:

- ▶ Timestamp String - time in human-readable form
- ▶ Timestamp Seconds - seconds from the epoch (January 1, 1970, 00:00 UTC)
- ▶ Event Kind - one of STARTUP, SHUTDOWN, LOGIN, LOGIN_FAIL, LOGOUT, or COMMIT_RESTORE.
- ▶ UserName - the UserProfile's userId, or "Stone" for the stone process.

- ▶ SessionId
- ▶ ProcessId
- ▶ Real UNIX user ID - numeric value
- ▶ Effective UNIX user ID - numeric value.
- ▶ Host Name - node name where the gem process is running.
- ▶ Gem IP Address - IP Address of the gem.
- ▶ Client IP Address - IP Address of the gem's client.
- ▶ NumCommits - number of commits performed by the session.
- ▶ Login UNIX user ID - numeric value
- ▶ KerberosPrincipal - the name of the principal used for passwordless login, or "".

STARTUP and SHUTDOWN records are written to indicate when the stone was started and stopped and do not indicate a session login or logout.

Login failures are written for non-exempt sessions that fail a login attempt, usually due to specifying a bad password.

Default: FALSE

STN_LOGIN_LOG_HALT_ON_ERROR

STN_LOGIN_LOG_HALT_ON_ERROR specifies the behavior of the stone if a write to the login log file fails. If this option is set to TRUE, the Stone will shutdown if the login log file cannot be written, due to lack of disk space or any other error. If this option is set to FALSE, the Stone does not halt but a warning message is printed to the Stone log.

Default: FALSE

STN_LOGIN_LOG_MAX_SIZE

STN_LOGIN_LOG_MAX_SIZE specifies the maximum size of the login log file in megabytes. Once the login log file reaches this size, it will be closed and a new login log file will be created. 0 means no file size limitation and the file size is not monitored by GemStone.

If no units are specified, the value is in KB. You may also specify units as KB, MB, or GB.

Default Units: MB

Default: 0 (no size limit)

Min: 0

Max: 1024 MB

STN_LOG_IO_FLAGS

STN_LOG_IO_FLAGS specifies what (if any) special I/O flags will be used to open the database transaction logs.

This configuration is only applicable for Solaris.

Direct I/O tells the operating system avoid caching extent data in the file system cache. Enabling direct I/O tells the operating system to treat the database tranlogs as if they were on raw partitions. Direct I/O may greatly improve database performance in some cases.

STN_LOG_IO_FLAGS has no effect for tranlogs on raw partitions.

STN_LOG_IO_FLAGS has the following possible values:

0 - no special I/O flags. This is the default.

1 - enable Direct I/O on all tranlogs on file systems (Solaris only).

If the requested I/O mode is not available, the stone will fail to start and an error message will be printed in the stone log. If this happens, change this option back to zero and restart the stone.

STN_LOG_IO_FLAGS has no effect on tranlogs which reside on raw partitions.

Min: 0

Max: 1 (Solaris only), 0 (All Others)

Default: 0

STN_LOG_LOGIN_FAILURE_LIMIT

If a user has greater than or equal to the STN_LOG_LOGIN_FAILURE_LIMIT number of login failures within the time in minutes specified by STN_LOG_LOGIN_FAILURE_TIME_LIMIT, a message is written to the Stone log file.

Changes to the runtime parameters require the OtherPassword privilege.

Runtime parameter: **#StnLogLoginFailureLimit**

Units: login attempts

Min: 0

Max: 65536

Default: 10

STN_LOG_LOGIN_FAILURE_TIME_LIMIT

If a user has greater than or equal to the STN_LOG_LOGIN_FAILURE_LIMIT number of login failures within the time in minutes specified by STN_LOG_LOGIN_FAILURE_TIME_LIMIT, a message is written to the Stone log file.

Changes to the runtime parameters require the OtherPassword privilege.

Runtime parameter: **#StnLogLoginFailureTimeLimit**

Units: Minutes

Min: 1

Max: 1440 (24 hours)

Default: 10

STN_LOOP_NO_WORK_THRESHOLD

STN_LOOP_NO_WORK_THRESHOLD indicates the maximum number of times the stone will continue executing its main service loop when there is no work to do. If the stone loops more than this number of times and finds no work, the stone will sleep for up to one second. The stone will immediately wake up when there is any work to be done.

Setting this value to zero disables this feature. Setting this value to a non-zero setting, in addition to causing the above behavior, will also cause the stone to not sleep whenever any of the following conditions are true and the no work threshold has not been exceeded:

- ▶ a session holds the commit token
- ▶ one or more sessions are waiting in the commit queue
- ▶ one or more sessions are waiting in the run queue.

Setting this parameter to a non-zero value will always cause the stone to consume more CPU.

Runtime parameter: **#StnLoopNoWorkThreshold**
Default: 0
Min: 0
Max: 536870911

STN_MAX_AIO_RATE

STN_MAX_AIO_RATE specifies the maximum I/O rate that each AIO page server thread is allowed when performing asynchronous writes. Since the I/O rate specified is applied to each thread, the total maximum I/O rate on the disk system is this value multiplied by STN_NUM_LOCAL_AIO_SERVERS.

The page server uses this maximum I/O rate for both dirty page and checkpoint writes.

Runtime parameter: **#StnMntMaxAioRate**
Min: 20
Max: 1000000
Default: 3000

STN_MAX_AIO_REQUESTS

STN_MAX_SESSIONS specifies the maximum number of asynchronous write requests the stone can have pending. If more than this number of asynchronous writes are requested, the stone will wait (sleep) until one or more of the pending requests have completed. Asynchronous write requests are only used to write to the current tranlog.

Min: (2 * STN_NUM_AIO_WRITE_THREADS)
Max: 4096
Default: the lower of SHR_PAGE_CACHE_NUM_PROCS and 128

STN_MAX_GC_RECLAIM_SESSIONS

The maximum number of page reclaim garbage collector sessions which are expected to be used on the system. When the default is specified, the actual value used is the number of extents defined in DBF_EXTENT_NAMES configuration.

Default: 0
Minimum: 0
Maximum: 256

STN_MAX_LOGIN_LOCK_SPIN_COUNT

Maximum number of times a session will attempt to write lock its user security data object at login time before raising a fatal error and failing the login. The session will sleep for 100 milliseconds between retries.

Enabling certain UserProfile security features (password aging, etc) causes each session to update its user security data object at login time and commit. A write lock must be acquired on this object to guarantee the commit succeeds.

Each UserProfile has a unique user security data object. Lock retries may be required when 2 sessions attempt to login with the same user ID at nearly the same instant. Simultaneous logins that user different user IDs never require lock retries.

Repositories that do not enable UserProfile security features are not affected by this parameter because the write-lock and commit described above are not required at login time.

Runtime parameter: **#StnMaxLoginLockSpinCount**

Default: 100

Minimum: 1

Maximum: 36000

STN_MAX_REMOTE_CACHES

STN_MAX_REMOTE_CACHES specifies the maximum number of remote shared page caches that the system may have.

Min: 0

Max: 10000

Default: 255

STN_MAX_SESSIONS

STN_MAX_SESSIONS limits the number of simultaneous sessions (number of Gem logins to Stone). The actual value used by Stone is the value of this parameter or the number of sessions specified by the software license key file, whichever is less. Using a value that is larger than needed will result in wasted space in the cache.

The number of logins may also be limited by changes in SHR_PAGE_CACHE_NUM_PROCS, or by the setting for the maximum number of file descriptors per process (imposed by the operating system kernel).

Recommended: 40 or more, depending on your requirements

Min: 1

Max: 32767

Default: 40

STN_MAX_VOTING_SESSIONS

STN_MAX_VOTING_SESSIONS specifies the maximum number of sessions that can simultaneously vote on possible dead objects, at the end of a markForCollection or epoch garbage collection. To help prevent the voting on possible dead objects from causing large increases in response time of the system, set this to a value substantially lower than STN_MAX_SESSIONS.

Runtime parameter: **#StnMaxVotingSessions**

Min: 1

Max: 1000000

Default: 100

STN_NUM_AIO_WRITE_THREADS

STN_NUM_AIO_WRITE_THREADS specifies the number of native threads the Stone will start to perform writes to the tranlog. In commit-intensive systems, this should be increased, generally to 8 or 16.

Cache Statistic: **StnAioNumWriteThreads** (Stone)

Min: 4

Max: 256
Default: 4

STN_NUM_GC_RECLAIM_SESSIONS

STN_NUM_GC_RECLAIM_SESSIONS specifies the number of page reclaim garbage collector sessions (Reclaim Gem sessions) that will be started when the Stone starts. This value must be less than or equal to STN_MAX_GC_RECLAIM_SESSION.

Runtime parameter: **#StnNumGcReclaimSessions**
Default: 1
Min: 0
Max: 256

STN_NUM_LOCAL_AIO_SERVERS

STN_NUM_LOCAL_AIO_SERVERS is the number of threads to be started in the AIO page server process on the Stone's machine. AIO page server threads write dirty pages from the shared page cache to the repository extents on disk.

For systems with two or more extents, in which the disk drive hardware allows concurrent writes to disk, multiple AIO page server threads can improve performance. You may not specify a number of AIO page server threads greater than the number of extents.

Min: 1
Max: 256
Default: 4 or the number of extents, whichever is smaller

STN_OBJ_LOCK_TIMEOUT

STN_OBJ_LOCK_TIMEOUT specifies the time in seconds that a session is allowed to wait to obtain one of the special single object write locks. For more information, see `System >> waitForApplicationWriteLock:queue:autoRelease:.`

Runtime parameter: **#StnObjLockTimeout**
Min: 0
Max: 86400
Default: 0 (stone waits forever)

STN_PAGE_MGR_COMPRESSION_ENABLED

STN_PAGE_MGR_COMPRESSION_ENABLED determines if the page manager thread in Stone will compress the list of pages that it sends to remote shared page caches for removal. If set to TRUE, all lists of pages larger than 50 will be compressed before transmission using the `> LZ4_compress()` function from the LZ4 compression library.

The same compressed list is used to send to all remote shared page caches; i.e., the compression operation is performed no more than once for each list of pages to be sent.

This option has no effect on systems that do not use remote shared page caches.

Runtime parameter: **#StnPageMgrCompressionEnabled**
Cache Statistic: **PageMgrCompressionEnabled** (Stone)
Default: FALSE

STN_PAGE_MGR_MAX_WAIT_TIME

Maximum time the Stone will defer servicing the page manager thread because the Stone is busy with other tasks. Normally the Stone services the page manager thread whenever it has idle time and no session is performing a commit. If the time the page manager thread has been waiting for service exceeds this value, the stone will service the page manager unconditionally and increment the cache statistic PageManagerStarvedCount.

Runtime parameter: **#StnPageMgrMaxWaitTime**
Units: Milliseconds
Default: 200
Min: 1
Max: 1000

STN_PAGE_MGR_PRINT_TIMEOUT_THRESHOLD

STN_PAGE_MGR_PRINT_TIMEOUT_THRESHOLD is the threshold in real seconds used by the page manager thread in Stone to determine if a slow response from a remote shared page cache should be printed to the page manager log file. If a remote cache takes longer than this number of seconds to respond to the page manager, the page manager will print a message to the log file. If a remote cache takes less than this number of seconds to respond, no message is printed.

Note that this value controls the writing of log messages only. The connection to the remote cache will not be terminated by page manager unless STN_REMOTE_CACHE_PGSRV_TIMEOUT is exceeded.

Runtime parameter: **#StnPageMgrPrintTimeoutThreshold**
Cache Statistic: **PageMgrPrintTimeoutThreshold** (Stone)
Min: 0
Max: 3600
Default: 5

STN_PAGE_MGR_REMOVE_MAX_PAGES

STN_PAGE_MGR_REMOVE_MAX_PAGES sets the maximum batch size for the Page Manager thread in Stone. This is the maximum number of pages in a single request to the stone. Must be greater than or equal to STN_PAGE_MGR_REMOVE_MIN_PAGES

Runtime parameter: **#StnPageMgrRemoveMaxPages**
Cache Statistic: **PageMgrRemoveMaxPages** (Stone)
Min: 1
Max: 16384
Default: 16384

STN_PAGE_MGR_REMOVE_MIN_PAGES

STN_PAGE_MGR_REMOVE_MIN_PAGES sets the minimum batch size for the Page Manager gem. When the number of pages waiting to be processed by the Page Manager is greater than this value, then the Page Manager will request the pages from the stone and process them. Otherwise the Page Manager will wait until this threshold is exceeded before requesting pages from the stone. Must be less than or equal to STN_PAGE_MGR_REMOVE_MAX_PAGES

Runtime parameter: **#StnPageMgrRemoveMinPages**
 Cache Statistic: **PageMgrRemoveMinPages** (Stone)
 Min: 0
 Max: 1792
 Default: 40

STN_PGSVR_PORT_RANGE

A list of two integers specifying the minimum and maximum port range to be used for listening sockets in pgsvr processes on the stone machine. Such pgsvr processes will listen on the first available port from this range, using the address that was used by the remote gem when it contacted the netldi to start the pgsvr, i.e. one of the addresses from the -A argument to that netldi's startnetldi command.

Both elements on the list must be between 1 and 65535, and the second element must be greater than the first.

Default: empty (random ports above 10000 will be used)

STN_PRIVATE_PAGE_CACHE_KB

STN_PRIVATE_PAGE_CACHE_KB sets the default size of the Stone page cache.

If no units are specified, the value is in KB. You may also specify units as KB, MB, or GB.

Default units: KB
 Min: 128 KB
 Max: 524288 KB
 Default: 2000 KB

STN_REMOTE_CACHE_PGSRV_TIMEOUT

STN_REMOTE_CACHE_PGSRV_TIMEOUT specifies the maximum time in seconds that the page manager session will wait for a response from a page server on a remote shared page cache. If no response is received within the timeout period, all Gems attached to that cache are logged off and a message is written to the Stone and page manager logs. Negative timeouts are not allowed. A timeout value of zero causes the page manager to wait forever.

Runtime parameter: **#StnRemoteCachePgsvrTimeout**
 Cache Statistic: **PageMgrRemoteCachePgsvrTimeout** (Stone)
 Min: 0
 Max: 3600
 Default: 15

STN_REMOTE_CACHE_STARTUP_TIMEOUT

STN_REMOTE_CACHE_STARTUP_TIMEOUT is the time in seconds allowed for startup of a remote shared cache. This is time from when first Gem on the remote machine connects to the Stone process until the remote cache completes its connection to the pagemanager thread in the Stone. Within this interval the first Gem forks the remote cache and that cache has to create its shared memory.

A scaledTimeout is also computed based on size of a shared cache

```
scaledTimeout := cacheSize <= 32GB
  ifTrue:[ (cacheSize / 32GB) * 120]
  ifFalse:[ 120 + ((cacheSize - 32GB) / 268GB) * 300 ]
```

If the scaledTimeout exceeds the configured value of STN_REMOTE_CACHE_STARTUP_TIMEOUT , the scaledTimeout is used for that cache.

Runtime equivalent: #StnRemoteCacheStartupTimeout
Default: 60
Min: 10
Max: 1800

STN_REMOTE_CACHE_TIMEOUT

STN_REMOTE_CACHE_TIMEOUT sets the time in minutes after the last active process on a remote node logs out before the Stone shuts down the shared page cache on that node.

A value of 0 causes the Stone to shut down the remote cache as soon as possible.

The runtime parameter can be changed only by SystemUser.

Runtime parameter: #StnRemoteCacheTimeout
Min: 0
Max: 5000000
Default: 5

STN_SHR_TARGET_PERCENT_DIRTY

STN_SHR_TARGET_PERCENT_DIRTY specifies the maximum percentage of the Stone's shared page cache that can contain dirty pages without AIO page server increasing its IO rates.

Runtime parameter: #StnShrPcTargetPercentDirty
Min: 1
Max: 90
Default: 20

STN_SIGNAL_ABORT_AGGRESSIVE

If the number of outstanding commit records exceeds STN_SIGNAL_ABORT_CR_BACKLOG, then STN_SIGNAL_ABORT_AGGRESSIVE is the maximum number of sessions which will receive SignalAbort when those sessions reference a commit record beyond STN_SIGNAL_ABORT_CR_BACKLOG and newer than the oldest commit record.

Changing the runtime parameter requires GarbageCollection privilege.

Runtime equivalent: #StnSignalAbortAggressive
Default: 0
Min: 0
Max: 500

STN_SIGNAL_ABORT_CR_BACKLOG

STN_SIGNAL_ABORT_CR_BACKLOG sets the number of old transactions (commit records) above which the Stone will start to generate SignalAbort messages to Gems. The Gem receives these as a TransactionBacklog exception.

If the Gem is not in transaction, this is received if the Gem has enabled receipt of sigAborts by invoking System >> enableSignaledAbortError. If the Gem that is out of transaction does not respond within the time allowed by STN_GEM_ABORT_TIMEOUT, the Gem will receive a ABORT_ERR_LOST_OT_ROOT.

If the Gem is in transaction, it will receive finishTransaction if it has invoked System >> enableSignaledFinishTransactionError. No further signals are sent to a Gem that is in transaction, whether or not it responds to the signal.

You may need to tune this option according to your application's commit rate and repository free space.

Changing the runtime parameter requires GarbageCollection privilege.

Runtime parameter: **#StnSignalAbortCrBacklog**
 Default: 20
 Min: 2
 Max: 65536

STN_SMC_SPIN_LOCK_COUNT

STN_SMC_SPIN_LOCK_COUNT specifies the number of times a process (gem or page server) that is waiting for a response from the stone will check a variable in shared memory before sleeping on a semaphore. Higher values can lead to higher throughput on performance-intensive systems, at the expense of higher CPU consumption.

Setting this value to 0 disables this feature, which causes the client process to always wait on the semaphore without checking the variable in shared memory.

Runtime equivalent: **#StnSmcSpinLockCount**
 Cache Statistic: **StnSmcSpinLockCount** (Stone)
 Min: 0
 Max: 10000000
 Default: 5000

STN_STATMONITOR_ARGS

STN_STATMONITOR_ARGS provides a way to automatically start statmonitor when the stone is started. If it is set to a non-empty list of arguments, these argument are passed to the statmonitor executable and statmonitor will be started automatically by the stone when it starts.

The list of possible arguments can be obtained by invoking statmonitor with only the "-h" argument. The first argument to statmonitor (the name of the stone) is automatically passed to statmonitor and must not be included here.

For example, to start statmonitor with a 5 second sample interval, with a date and timestamp included in the output file name, and with the resulting output file placed in the directory /statmonFiles, the following setting would be configured:

```
STN_STATMONITOR_ARGS =
"-i5 -u5 -d /statmonFiles -F'statmon_%%S_%%P_%d-%m-%y-%H:%M:%S'";
```

Default: "" (statmonitor will not be started)

STN_STONE_CACHE_STARTUP_TIMEOUT

STN_STONE_CACHE_STARTUP_TIMEOUT specifies the time in seconds allowed for the startup of the shared cache on the Stone's machine. This is time from when the Stone forks the page cache monitor process until it is able to complete its connection to that process.

A scaledTimeout is also computed based on the size of the shared cache

```
scaledTimeout := cacheSize <= 32GB
  ifTrue:[ (cacheSize / 32GB) * 120]
  ifFalse:[ 120 + ((cacheSize - 32GB / 268GB) * 300) ]
```

If the computed scaledTimeout exceeds the configured value of STN_STONE_CACHE_STARTUP_TIMEOUT, the scaledTimeout is used.

Min: 10
Max: 1800
Default: 60

STN_SYMBOL_GC_ENABLED

STN_SYMBOL_GC_ENABLED determines if symbol garbage collection is allowed to run on the system. Setting this value to true enables symbol garbage collection.

Updating the runtime parameter requires the GarbageCollection privilege.

Runtime parameter: **#StnSymbolGcEnabled**
Default: FALSE

STN_SYMBOL_GEM_TEMPOBJ_CACHE_SIZE

STN_SYMBOL_GEM_TEMPOBJ_CACHE_SIZE specifies the temporary object memory size of the Symbol Gem. The Stone's runtime parameter #StnSymbolGemTocSize is initialized from STN_SYMBOL_GEM_TEMPOBJ_CACHE_SIZE at stone startup. If the Symbol Gem dies (other than from an explicit execution of System stopSymbolGem), and is automatically restarted, then the setting for #StnSymbolGemTocSize setting will be doubled.

This doubling only occurs for the first restart since either Stone startup or an runtime change to #StnSymbolGemTocSize.

While the default is normally 20MB, if repository conversion is detected at stone startup (startstone -C) the default is 200MB.

Runtime equivalent: **#StnSymbolGemTocSize**
Min: 10000KB
Max: 10GB
Default: 20MB

STN_TRAN_FULL_LOGGING

If STN_TRAN_FULL_LOGGING is set to true, all transactions are logged, and log files are not deleted by the system. This is full transaction logging mode. In this mode, the transaction logs are providing real-time incremental backup of the repository. If no disk space is available for logs, Gem session processes may appear to "hang" until space becomes available.

If STN_TRAN_FULL_LOGGING is set to false, only transactions smaller than STN_TRAN_LOG_LIMIT are logged; larger transactions cause a checkpoint, which updates the extent files. This is partial transaction logging mode. Log files are deleted by the system when the circular list of log directories wraps around. This setting allows a simple installation to run unattended for extended periods of time, but it does **not** provide real-time backup.

Once you have started the Stone on a repository with `STN_TRAN_FULL_LOGGING = true`, then the true state will persist in the repository; any subsequent changes to this parameter in the configuration file are ignored. To change the repository back to partial logging, you must do a Smalltalk full backup and then restore the backup into a copy of `$GEMSTONE/bin/extent0.dbf`.

For further information, see Chapter 10, “Managing Transaction Logs”.

Default: NONE. The system will not run unless you specify the logging type.

STN_TRAN_LOG_DEBUG_LEVEL

This option is only for GemStone internal use. Customers should not change the default setting unless directed to do so by GemStone Technical Support.

Runtime parameter: `#StnTranLogDebugLevel`

Default: 0

STN_TRAN_LOG_DIRECTORIES

`STN_TRAN_LOG_DIRECTORIES` lists the directories or raw disk partitions to be used for transaction logging. This list defines the maximum number of log files that will be online at once. Each entry must be a directory or a raw disk partition. Directories may appear multiple times in the list. A given raw disk partition may appear only once. If raw partitions are used or if `STN_TRAN_FULL_LOGGING` is false, at least two entries should be included.

`STN_TRAN_LOG_DIRECTORIES` may be set to `/dev/null` in development systems; this means that no tranlog records at all are written. If the repository shuts down unexpectedly, committed work done since the last checkpoint is irretrievably lost.

Min: 1 entry

Max: 2^{31} entries

Default: Empty (the system will not run without at least one entry)

Initial setting: `$GEMSTONE/data/`

STN_TRAN_LOG_LIMIT

`STN_TRAN_LOG_LIMIT` sets the maximum transaction log entry size limit in KB. Successful commits of transactions consuming more than this amount of log file space when `STN_TRAN_FULL_LOGGING` is set to false will cause a checkpoint. This option has no effect when `STN_TRAN_FULL_LOGGING` is set to true.

If no units are specified, the value is in KB. You may also specify units as KB, MB, or GB.

The runtime parameter can be changed only by SystemUser.

Default units: KB

Runtime parameter: `#StnTranLogLimit`

Min: 25 KB

Max: 1000 KB

Default: 1000 KB

STN_TRAN_LOG_PREFIX

`STN_TRAN_LOG_PREFIX` sets file name prefixes for transaction log files. A sequence number and `.dbf` is added to the prefix; for example, `“tranlog”` produces

“tranlog0.dbf, tranlog1.dbf, ...”. You can set this configuration option to permit multiple repository monitors to share a log directory without conflict.

Default: tranlog

STN_TRAN_LOG_SIZES

STN_TRAN_LOG_SIZES sets the maximum sizes of all transaction log files, in order and separated by commas. Each size applies to a corresponding log file specified in STN_TRAN_LOG_DIRECTORIES, and the number of entries must match.

If no units are specified, the value is in MB. You may specify units as KB, MB, or GB.

Default units: MB

Min: 10 MB

Max: 16384 MB

Default: Empty (the system will not run unless sizes are specified)

Initial setting: 100 MB

STN_TRANQ_TO_RUNQ_THRESHOLD

The number of sessions in the commit queue (waiting for the commit token) that are allowed to simultaneously process unions (read old commit records) while waiting for the commit token.

For example, if this parameter is set to 2, then sessions commitQueue[1], and commitQueue[2] (if they exist) will process unions. The first session in the commit queue, commitQueue[0], will never process unions since it will receive the token when the current commit completes.

Cache Statistic: (Stone) StnTranQToRunQThreshold

Runtime equivalent: #StnTranqToRunqThreshold (may only be set by SystemUser)

Default: 2

Min: 1

Max: 20

STN_WELL_KNOWN_PORT_NUMBER

STN_WELL_KNOWN_PORT_NUMBER is the port number that the Stone will use as its well-known port. The well-known port is used by all Gems while establishing their initial connection to the Stone during the login sequence.

If the specified port is in use by another process, the Stone will not start and exits with an error.

A value of zero indicates the port number will be selected by the system.

Min: 1

Max: 65535

Default: 0

A.4 Runtime-only Configuration Options

The parameters described in this section are similar to the configuration options above, but can only be read or modified at runtime.

The process for modifying is similar to that for the runtime parameter equivalents for the configuration options listed in the configuration files.

The runtime parameters are read using the method `System class>>configurationAt:`, and updating using `System class>>configurationAt:put:.`

GemConvertArrayBuilder

If True, allows old style Array Builder syntax `#[a, b]` to be parsed correctly. The compiler converts this syntax to the new form `{ a . b }`, and updates method source as well as compiled code.

Default: false

GemDropCommittedExportedObjs

If this configuration parameter is true, clean, committed objects may be dropped from RAM. This reduces demand on memory in the Gem, but there is the small cost of an additional bitmap lookup when the object is faulted, to detect if this object is in the Pure Export Set.

Default: false

GemExceptionSignalCapturesStack

If this configuration parameter is true, invocations of `AbstractException>>_signalWith:` fill in the `gsStack` instance variable of the receiver, allowing subsequent calls to `Exception >> stackReport.`

Default: false

LogOriginTime

`#LogOriginTime` is the time the current sequence of Stone logs was started. It is the same value returned by `Repository>>logOriginTime.` For information about when a new sequence is started, see the method comment for `Repository>>commitRestore` in the image.

This should not be modified.

SessionInBackup

`#SessionInBackup` is the GemStone session number of the session performing a full backup, or -1 if a backup is not in progress.

This should not be modified by the user.

StnCurrentTranLogDirId

`#StnCurrentTranLogDirId` is the one-based offset of the current transaction log into the list of log directory names, `STN_TRAN_LOG_DIRECTORIES`. It is the same value returned by `Repository>>currentLogDirectoryId`.

This should not be modified.

StnCurrentTranLogNames

`#StnCurrentTranLogNames` is an Array containing up to two Strings: the name of the transaction log to which records currently are being appended, and the name of the current replicated log. These are the same values returned by `Repository>>currentLogFile` and `currentLogReplicate`, respectively.

StnLogFileName

The stone log file path and name.

This is a read-only value.

StnLogGemErrors

`#StnLogGemErrors` is intended for internal debugging use. When it is set to 1, the Stone logs error messages it sends to Gems.

StnLoginsSuspended

`#StnLoginsSuspended` ordinarily has the values 0 (false) and 1 (true) as set by `System class>>suspendLogins` and `resumeLogins`.

Changing this parameter requires the `SystemControl` privilege.

StnMaxReposSize

`#StnMaxReposSize` is the maximum size of the repository for your GemStone license, as set by the GemStone keyfile. The value of this parameter is 0 when not limited by the keyfile.

This should not be modified.

StnMaxSessions

`#StnMaxSessions` is the maximum allowed number of sessions for your GemStone license, as set by the GemStone keyfile. It is not related to the `STN_MAX_SESSIONS` configuration option. The value of this parameter is 0 when not limited by the keyfile.

This should not be modified.

StnStandbyRole

`#StnStandbyRole` reflects the role of this stone in a hot standby system. It should not be modified.

StnSunsetDate

`#StnSunsetDate` is the sunset date for your GemStone license, as set by the GemStone keyfile. The value of this parameter is 0 when not limited by the keyfile.

This should not be modified.

StnTranLogOriginTime

`#StnTranLogOriginTime` is the time when the current transaction log was started.

This should not be modified.

GemStone Utility Commands

The GemStone/S 64 Bit utility commands in this appendix are provided in the `$GEMSTONE/bin` directory.

- ▶ **copydbf** (page 336) – copy an extent, transaction log or backup.
- ▶ **gslis**t (page 341) – list running GemStone server processes.
- ▶ **pageaudit** (page 344) – offline audit of repository pages.
- ▶ **pstack** (page 345) – get C-level and Smalltalk stack traces.
- ▶ **removedbf** (page 346) – delete an extent, transaction log or backup.
- ▶ **startcachewarmer** (page 347) – start cache warmers.
- ▶ **startlogreceiver** (page 349) – start logreceiver for hot standby system.
- ▶ **startlogsender** (page 351) – start logsender for hot standby system.
- ▶ **startnetldi** (page 353) – start a NetLDI.
- ▶ **startstone** (page 355) – start a Stone.
- ▶ **statmonitor** (page 357) – start logreceiver for hot standby system.
- ▶ **stoplogreceiver** (page 360) – stop logreceiver for hot standby system.
- ▶ **stoplogsender** (page 361) – stop logsender for hot standby system.
- ▶ **stopnetldi** (page 362) – shut down a NetLDI.
- ▶ **stopstone** (page 363) – shut down a Stone.
- ▶ **topaz** (page 364) – command line scripting tool for GemStone. See also the separate manual *Topaz Users's Guide*.
- ▶ **vsd** (page 366) – graphic tool to analyze GemStone statistics files. See also the separate manual *VSD Users's Guide*.
- ▶ **waitstone** (page 367) – verify status of a Stone.

copydbf

copydbf *sourceNRS destinationNRS* [-C | -z | -Z | -c | -u] [-E] [-f *filePrefix*]
 [-n *netLdiName*] [-p *pgsvrId*] [-s *Mbytes*] [-l | -m | -P]

copydbf *sourceNRS* -i | -I [-n *netLdiName*] [-p *pgsvrId*]

copydbf *sourceNRS*-W *encrCertFN* | -X *signCertFN* | -Y *sigFN*

copydbf -V *sourceNRS* [-K *keyRingDirs*]

copydbf -h | -v

- sourceNRS* The source file or raw partition (containing an extent, a transaction log, or a backup file created by `fullBackupTo` :) as a GemStone network resource string.
- destinationNRS* The destination file, directory, or raw partition as a GemStone network resource string. If the destination is a file system directory (the trailing / is optional), a file name is generated and appended to *destinationNRS* based on the type and internal fileId of the source. Use of /dev/null as the destination is supported only for files as a means of verifying that the file is readable.
- c Write the transaction log with individual record-level compression. This option only applies to transaction logs. Used by hotstandby utilities.
- C Write the entire output file compressed, in gzip format; the same as the -z option. The output must be a filesystem file, and you must specify the .gz in the filename.
- E Ignore disk read errors. If the disk read error occurs while reading an extent root page, the copy will fail. Otherwise, pages of the source file that cannot be read will be replaced with an invalid-page-kind page in the destination file. The destination file may not be usable. This option only applies to extents, not to transaction logs or backup files.
- f *filePrefix* If *destinationNRS* is a file system directory, then *filePrefix* overrides the filename prefix that would be generated based on the contents of *sourceNRS*. If *destinationNRS* is other than a file system directory, this option has no effect.
- h Print usage and exit.
- i Information only. When this option is present without *destinationNRS*, information about *sourceNRS* is printed without performing a file copy. If both -i and *destinationNRS* are present, an error message is printed.
- I Full information. The same information is printed as for -i. In addition, if the file is a transaction log, all checkpoint times found are listed instead of only the last one.

- K** *keyRingDirs* For secure backup. List of colon-separated directories which contain keys and certificates, which is used with the -V option to verify the digital signature.
- l** Least-significant-byte ordering for the *destinationNRS*. This byte ordering is the native byte ordering for Intel processors.
- m** Most-significant-byte ordering for the *destinationNRS*. This byte ordering is the native byte ordering for Solaris SPARC, AIX POWER, and HP-UX PA-RISC and Itanium processors.
- n** *netLdiName* The name of the GemStone network server; the default is `gs64ldi`.
- P** Preserve byte ordering. This option creates the destination file using the byte ordering found in the source file. The default is to write the file using the host's native byte ordering.
- p** *pgsvrId* The name of a specific `runpgsvr` (similar to `gemnetid`).
- s** *MB* The size (in MB) to pre-allocate the destination file. For instance, `-s10` allocates at least 10 MB to the created file. If you do not specify the `-s` option, the output file is made as short as possible.
- u** Uncompress output records (default is to preserve record-level compression from input); used by hotstandby utilities.
- v** Print version and exit.
- V** For secure backup. Verify the digital signature of a secure backup. Secure backup must be local. This requires including either the `-K` command line option, or setting the `GEMSTONE_KEYRING_TABS` environment variable.
- W** *encrCertFN* For secure backup. Create a new file or files with the given name containing the encryption certificate(s) from a secure backup to file(s). Secure backup must be local. Cannot be used with other write options on a single command line.
- X** *signCertFN* For secure backup. Create a new file with the given name containing the signing certificate from a secure backup to a file. Secure backup must be local. Cannot be used with other write options on a single command line.
- Y** *sigFN* For secure backup. Create a new file with the given name containing the digital signature from a secure backup to file. Secure backup must be local. Cannot be used with other write options on a single command line.
- z** Write the output file compressed, in `gzip` format; the same as the `-C` option. The output must be a filesystem file, and you must specify `.gz` in the filename.
- Z** Write the output file compressed, in `LZ4` format. The output must be a filesystem file, and you must specify `.lz4` in the filename.

To make copies of extent files or transaction logs, the user executing **copydbf** must have read permission to the file. If you attempt to copy extent files that are in use, if checkpoints are not suspended the resulting repository may be corrupt and unusable. See “How To Make an Extent Snapshot Backup” on page 212 for more information.

GemStone repository files on the UNIX file system or to an NFS-mounted drive can usually be copied using the ordinary `cp` command, as well as using **copydbf**.

If you are copying between operating systems and the source and destination have different byte ordering, you should use **copydbf**. You can use the **copydbf** for disk-to-disk copies between machines when the remote disk is not NFS-mounted; to use **copydbf** between remote nodes in this case, you must have a NetLDI running on both nodes, and authentication must be configured to allow access.

If the destination is a directory in a file system, **copydbf** generates a filename based on the type of file. The generated name includes a prefix (`extent`, `tranlog`, or `backup`), a `fileId` representing an internal sequence number that starts at 0, and the extension `.dbf`. You can use the `-f` option to change the prefix.

A message describing the source and destination files is printed to `stderr` before starting the copy. The size of the destination file is printed to `stderr` after the copy is completed.

For example:

```
% copydbf $GEMSTONE/bin/extent0.dbf .
```

```
Source file: /users/GemStone/data/extent0.dbf
  File type: extent  fileId: 0
  ByteOrder: Intel (LSB first)  compatibilityLevel: 844
  Last checkpoint written at: 08/27/2017 19:03:46 PDT.
Destination file: ./extent0.dbf
  ByteOrder: Intel (LSB first)
  Clean shutdown, no tranlog needed for recovery.
  Last tranlog written to had fileId 5 ( tranlog5.dbf ).
  File contains 4608 records occupying 75.5 MBytes.
```

Compression of copydbf output

The output of **copydbf** can be compressed at the file level using `gzip` or `LZ4` compression (`-C|-z|-Z`), or compressed at the record level. (`-c`). `Tranlog restore` and **copydbf** can read any of the compression variants.

Record-level compression is used by the `hotstandby` implementation. The `logsender` does record-level compression before sending `tranlog` records to the log receiver. The transmitted `tranlogs` that are written on the slave system by the `logreceiver` are record-level compressed. File-level `zip/LZ4` compressed `tranlogs` are converted into record-level compressed `tranlog` records by the `logsender`.

Using copydbf to access information on a file

To obtain the same source file information (but not the size) without making a copy, use the second form of the command: **copydbf -i sourceNRS**. In this usage, you do not specify a *destinationNRS*.

The following **copydbf -i** example displays information for an extent, and indicates the oldest transaction log that would be needed to recover from a system crash:

```
% copydbf -i extent0.dbf
```

```
Source file: extent0.dbf
  File type: extent  fileId: 0
  ByteOrder: Intel (LSB first)  compatibilityLevel: 844
  Last checkpoint written at: 08/04/2017 19:03:46 PDT.
  Oldest tranlog needed for recovery/restore is fileId 5
    ( tranlog5.dbf ).
  Extent was shutdown cleanly; no recovery needed.
```

The next example displays information for a backup, and indicates the oldest transaction log that would be needed to restore subsequent transactions.

```
% copydbf -i backup.dat
```

```
File type: backup  fileId: 0
  ByteOrder: Intel (LSB first)  compatibilityLevel: 850
  The file was created at: 08/27/2017 19:03:46 PDT.
  Full backup started from checkpoint at: 08/27/2017 19:03:46 PDT.
  Oldest tranlog needed for restore is fileId 2 ( tranlog2.dbf ).
  Backup was created by GemStone Version: 3.2.0 .
```

To obtain the size of a repository file in a raw partition, use this form:

```
% copydbf sourceNRS destinationNRS
```

For a listing of all checkpoints recorded in a transaction log, use **copydbf -I sourceNRS**. This information is helpful in restoring a GemStone backup to a particular point in time. For example:

```
% copydbf -I tranlog5.dbf
```

```
Source file: tranlog5.dbf
  File type: tranlog  fileId: 5
  ByteOrder: Intel (LSB first)  compatibilityLevel: 933
  The file was created at: 08/27/17 11:49:05 PST.
  The previous file last recordId is 36 .
  Scanning file to find last checkpoint...
  Checkpoint 1 started at: 08/27/17 11:54:35 PST.
    oldest transaction references fileId -1 ( this file ).
  Checkpoint 2 started at: 08/27/17 11:56:17 PST.
    oldest transaction references fileId -1 ( this file ).
  Checkpoint 3 started at: 08/27/17 11:58:30 PST.
    oldest transaction references fileId -1 ( this file ).
  Checkpoint 4 started at: 08/27/17 12:03:01 PST.
    oldest transaction references fileId -1 ( this file ).
  File contains 69 records occupying 35328 bytes.
```

copydbf with remote nodes

You must give an NRS (network resource string) for both the source file and the destination. A local machine file specification is a valid NRS.

In the following example, the local GemStone repository file “extent0.dbf” is copied to a remote machine using a full *destinationNRS*. In this example, the repository file is copied to a remote machine named “node,” using remote user account “username” and “password,” with a remote filespec of “/users/path/extent0.dbf_copy,” via the standard GemStone network server *gs641di* using TCP protocol:

```
% copydbf $GEMSTONE/data/extent0.dbf
  \!@node#auth:username@password#dbf\!/users/extent0.dbf_copy
```

copydbf with raw partitions

copydbf of backup files to raw partitions is not supported.

The next example copies a fresh repository extent to an existing raw disk partition. If the raw partition already contains a repository file or backup, use **removedbf** first to mark it as being empty.

```
% copydbf $GEMSTONE/bin/extent0.dbf /dev/rsd3h
```

Copying a transaction log from a raw partition to a file system directory generates a file name having the same form as if that transaction log had originated in a file system. If the internal fileid is 43, this example would name the destination file

```
/dsk1/tranlogs/tranlog43.dbf:
```

```
% copydbf /dev/rsd3h /dsk1/tranlogs/
```

gslist

```
gslist [ -c ] [ -l | -p | -x ] [ -m host ]+ [ [ -n ] name ]+ [ -N netldiName ] [ -q ] [ -s key | -S key ]  
      [ -t secs ] [ -u user ] [ -v ]
```

```
gslist -h | -V
```

- | | |
|----------------------|--|
| -c | Removes locks left by servers that have been killed. |
| -h | Print usage and exit. |
| -l | Prints a long listing (includes pid and port). |
| -m <i>host</i> | Only list servers on machine <i>host</i> ; default is '.' which represents the local host. If not the local host, the machine <i>host</i> must be running a compatible version of NetLDI, and the -N argument is required if it is not running with the default name in the environment of gslist. |
| -n <i>name</i> | Only list the server <i>name</i> . |
| -N <i>netldiName</i> | When -m is used, -N specifies name of netldi to contact on remote host. Default is name specified in GEMSTONE_NRS_ALL environment variable, otherwise name is 'gs64ldi'. |
| -p | Prints only the pid (process id), or 0 if the server does not exist. |
| -q | (Quiet.) Don't print any extra information; intended for use when the output will be processed by some other program. |
| -s <i>key</i> | Sort results by the given sort key. Legal key values:

name sort by process name (default).
owner sort by process owner.
time sort by process start time.
type sort by process type.
version sort by process version. |
| -S <i>key</i> | Same as -s except results are sorted in reverse order. |
| -t <i>secs</i> | Wait <i>secs</i> seconds for server to respond (only with -v); default is 2 seconds. |
| -u <i>user</i> | Only list servers started by <i>user</i> . |
| -v | Verify the status of each server. |
| -V | Print the version information and exit. |
| -x | Prints an exhaustive listing, with each item on a separate line. |

The **gslist** command prints information about GemStone servers. The default listing prints the following server attributes in columns:

Status	One of the following: contRestore Continuous restore for hot standby (-v only) exists server process exists but is not verified frozen server is not responding (-v only) full server can't accept any more clients (-v only) killed server process does not exist OK server is accepting clients (-v only) restoreBkup server is in restore from backup (-v only) restoreLogs server is in restore from logs mode (-v only) recovery Restart after unclean shutdown in progress (-v only) startup server process is not yet accepting clients (-v only)
Version	GemStone version of the server.
Owner	The account name of the user who created the server.
Started	The date and time that the server was started.
Type	One of the following: Netldi, Stone, cache (shared page cache monitor), logsender, logreceiver
Name	The server's name.

When you include the **-l** or **-x** option, the following additional columns are printed:

Pid	The process id of the server's main process.
Port	The port number of the server's listening socket.

The **-x** option prints the preceding attributes on separate lines, and adds lines for the following as appropriate:

options	Options used when the server was started.
logfile	Full path of server's log file, if it exists.
sysconf	The GemStone system configuration file. See "System Configuration File" on page 286.
execonf	The GemStone executable configuration file. See "Executable Configuration File" on page 287.
GEMSTONE	Root of the product tree used by the server.

If many servers are reported as **frozen**, try increasing **-t secs**.

By default, status is returned for all servers on the current host. To specify a particular server, use the **-n** switch, or just include the server name on the command line (since the **-n** is optional). To specify multiple server names, include the **-n name** option for each server.

Remote queries

The **-m** option allows you to list servers on a remote host. To specify more than one remote host, include the **-m host** option multiple times, one for each host. Names on remote hosts are prefixed by `host :`, where *host* is the name of the remote machine.

To list servers on a remote host, **gslis**t must be able to contact a NetLDI running on the remote machine.

- ▶ The remote NetLDI must be a GemStone version that is compatible with the version of **gslis**t.

This limitation applies to the remote NetLDI, but not to the Stone and other server processes on the remote host, for which **gslis**t fetches information.

- ▶ The remote NetLDI must be named with the default name, or the **-N** argument should be included to specify the name.

The default name here is the default for the environment in which **gslis**t is running. This is either "gs64ldi", or a name specified by `$GEMSTONE_NRS_ALL`.

Date and Time format

The date and time that the process started is normally printed in a format specific to **gslis**t and fitting into the table display. To get a parse-able but potentially less readable format, you may use the environment variable `GS_GSLIST_TIME_FORMAT` to specify a UNIX-style date format string.

Exit status

The exit status has the following values:

- 0 Operation was successful.
- 1 No servers were found.
- 2 A stale lock was removed (in response to **-c** switch).
- 3, 4 An error occurred.

gslis on Windows

gslist is available for Windows, in addition to regular server platforms, as part of the Windows Client installation. **gslis**t on Windows must be used with the **-m** option, specifying a hostname for a host that is running GemStone.

pageaudit

pageaudit [*gemStoneName*] [**-e** *exeConfig*] [**-z** *systemConfig*] [**-f**] [**-d**] [**-l** *logfile*] [**-h**]

<i>gemStoneName</i>	Name of the GemStone repository monitor; the default is <code>gs64stone-audit</code> . Network resource syntax is not permitted.
-d	Disable audit of data pages. Only audit Object Table pages, bitmaps, and other non-data pages.
-e <i>exeConfig</i>	The GemStone executable configuration file. See “Executable Configuration File” on page 287.
-f	Keeps running beyond the first error, if possible
-h	Print usage and exit.
-l <i>logfile</i>	Write output to the file with the given name. The file is created if it does not exist. If there is an existing file with this name, the <code>pageaudit</code> output is appended to the end of this file.
-n <i>numSessions</i>	Run with the specified number of sessions. The default is <code>numExtents + numCpus</code> .
-z <i>systemConfig</i>	The GemStone system configuration file. See “System Configuration File” on page 286.

Audit the pages in a GemStone repository, which must not be in use. **pageaudit** opens the repository specified by the relevant configuration files. The arguments **-e** *exeConfig*, and **-z** *systemConfig* determine which configuration files **pageaudit** reads.

pageaudit by default runs aggressively, using a large number of threads (based on the number of CPUs and extents for this system), in order to complete as quickly as possible; the **-n** option can be used to specify fewer threads, and reduce the impact on your system.

By default, all pages in the repository are verified; this includes data pages as well as Object Table, bitmap, and other pages containing internal information. Audit of data pages can be disabled using the **-d** option.

An error is returned if another Stone is running as *gemStoneName* or has opened the same repository.

When you include the **-f** switch, **pageaudit** prints all errors possible. Without **-f**, the default is to stop after the first error is found.

This utility can take a long time to run, so it is best to run it as a background job.

For additional information about **pageaudit** and a description of its output, see “Page Audit” on page 132.

pstack

pstack [**-b** | **-c** | **-C** | **-h**] *processPid*

<i>processPid</i>	The PID of a running GemStone process.
-b	Print brief C and Smalltalk stacks.
-C	Print brief C stack, omit Smalltalk stack.
-c	Print brief and full C stack, omit Smalltalk stack.
-h	Print usage and exit.

The **pstack** command attaches a debugger to the process with the given pid, outputs the C and Smalltalk stacks of that process to stdout, and detaches. The Smalltalk stack is printed to the Gem log or topaz linked session console.

Execution of the running process briefly pauses while the debugger is attaching, but the process will subsequently continue running normally.

pstack is similar to functions provided with some OS platforms.

By default, the stack summary is printed with one line per frame for each thread, followed by the complete stack details. The **-b** and **-C** options allow you to specify brief format stacks.

By default, both C stack and (if there is a Smalltalk context) Smalltalk stacks are printed. The **-c** and **-C** options allow you to omit the Smalltalk stack.

removedbf

removedbf *dbfNRS* [**-h**]

dbfNRS The GemStone repository filename or device, as a network resource string, for the repository to be removed.

-h Print usage and exit.

This command removes (erases) a GemStone repository file. It is provided primarily for erasing an extent or transaction log from a raw partition, but it also works on files in the file system. It does not work for disks on remote file systems.

If you specify a file in the file system, this command is equivalent to the **rm** command. If you specify a raw disk partition, GemStone metadata in the partition is overwritten so that GemStone will no longer think there is a repository file on the partition.

For example, to remove an extent on the raw partition `/dev/rsd3h`:

```
removedbf /dev/rsd3h
```

This command does not disconnect an extent from the logical repository. To alter the configuration by disconnecting an existing extent, see “To Remove an Extent” on page 189.

startcachewarmer

```
startcachewarmer [ -d | -D ] [ -h ] [ -l limit ] [ -L path ] [ -n numSessions ] [ -s stoneName ]
  [ -w writeInterval ] [ -W ] [ -C midCacheSizeKb ] [ -e gemConfigPath ] [ -H stoneHost ]
  [ -M midCacheHost ] [ -N midCacheMaxProcs ]
```

- C *midCacheSizeKb*** The size of the mid-level cache in KB (default 75000). Only used if the **-M** option is specified and the mid-level cache does not exist.
- d** Reads data pages into the cache (default: only object table pages are read).
- D** Reads data pages into the UNIX file buffer cache and not the shared page cache.
- e *gemConfigPath*** path to a gem config file, by default gem.conf in the current directory. Used to specify a shared cache configuration, if the warmer starts a remote or mid-level cache.
- h** Print usage and exit.
- H *stoneHost*** The host name or IP address where the stone is running. This option should only be used when warming a remote cache.
- l *cacheFullLimit*** Stops cache warming if the number of free frames in the cache falls below the specified *cacheFullLimit*. If *cacheFullLimit* is -1 (the default), have the system compute the actual limit based on the size of the shared cache. If *cacheFullLimit* is 0, force cache warming to continue even if the shared cache is full.
- L *path*** Path to a writable log file directory (default: current directory)
- M *midCacheHost*** The host name or IP address where the mid-level cache is running or will be created. The **-H** option must also be specified with this option (default: no mid-level cache is used)
- n *numSessions*** Number of worker sessions to start. The default is the number of CPUs + number of Extents, plus 1 additional master session. The cachewarmer will exit if not enough sessions are available.
- N *midCacheMaxProcs*** The maximum number of processes that can use the mid-level cache (default: 50). Only used if the **-M** option is also specified and mid-level cache does not exist.
- s *stoneName*** Name of the running Stone (default: gs64stone).

- W** Wait for cache warming Gems to exit before exiting this script. By default, this script spawns Gems in the background and exits immediately.
- w** *writeInterval* Instruct the shared page cache monitor to write the ids of all data pages in the shared cache to the working set file, at the given interval in minutes. A value of 0 means write the file only when the stone is shutdown or killed. The working set is written to `/opt/gemstone/locks/stoneName~hostid.workingSet.lz4`

The **startcachewarmer** command warms up the shared page cache on startup, by preloading object and optionally data tables into the cache. This allows the overhead of initial page loading to occur in a controlled way on system startup, rather than more gradually as the repository is in use.

The object table and dependency map pages are always loaded; data pages are loaded based on the **-d** flag and the presence of the working set file.

- ▶ If the working set file `/opt/gemstone/locks/stoneName~hostid.workingSet.lz4` exists, the valid data pages in this file are loaded.
- ▶ If the working set file does not exist, then behavior depends on the use of the **-d** and **-D** options:
 - ❑ If the **-d** option is specified, all data pages in page order are loaded.
 - ❑ With the **-D** option, data pages are loaded into the OS file buffer but not into the shared page cache.

The **-d** and **-D** flags are mutually exclusive, if both are specified, then the later one is used.

Cache warming writes messages to stone log when it starts and with status when it completes. When the **-W** option is specified, it will also write the results to the console. If an error occurs during cache warming, details are preserved in a file named `stonename_cachewarmer.log` in the directory from which this utility was executed.

If a remote shared page cache is to be warmed (i.e., the **-H** option is used), then the remote cache will be created if it does not already exist. The configuration used to start the remote cache are controlled by the **-e** argument, a `gem.conf` in the current directory, or the `GEMSTONE_EXE_CONF` environment variable.

If a mid-level cache host name or IP address is specified (via **-M**), the mid-level cache will be created if it does not already exist. The **-C** and **-N** options will be used to specify the size and number of processes that can attach the mid-cache respectively. If the mid-cache already exists, the **-C** and **-N** options are ignored.

For greater efficiency, you can set the configuration file parameters `STN_CACHE_WARMER_ARGS`, `GEM_CACHE_WARMER_ARGS`, AND `GEM_CACHE_WARMER_MID_CACHE_ARGS` to invoke cache warming automatically on startup.

startlogreceiver

```
startlogreceiver -P listeningPort -A listeningAddress -T tranlogDir+ [ -I logFile ]
    [ -s stoneName ] [ -p alternatePort ] [ -d ] [ -t timeoutSeconds ] [ -v ] [ -h ]
    [ -C certFileName ] [ -J certAuthFileName ] [ -K keyFileName ] [ -Q passphrasestring ] [ -S ]
    [ -V ]
```

- A** *listeningAddress* Address that will be used to attempt connection to a logsender.
 - d** Print debug tracing of tranlog read and write operations to log file. The log file will be much larger.
 - I** *logFileOrDir* The path and filename or directory for the logged output of the logreceiver process. The default is `/opt/gemstone/log/logreceiver_listeningPort.log`.
 - h** Print usage and exit.
 - P** *listeningPort* Port or named service that will be used to attempt connection to a logsender.
 - p** *alternatePort* The logreceiver listens on localhost on this port for stoplogreceiver commands. If logsender and logreceiver are run on same machine, then **-p** must be used to specify a port different than *listeningPort* specified with the **-P** argument.
 - s** *stoneName* The name of the slave stone. *stoneName* is required for the logreceiver to be able to notify a stone in continuous restore mode that new log records have arrived. Without *stoneName*, logreceiver will just write tranlogs to the file system.
 - T** *tranlogDir* Directory(s) where logreceiver writes files received from the logsender, and `continuousRestoreFromArchiveLogs`: will read from. At least one is required, up to 20 **-T** may be specified.
 - t** *timeoutSeconds* How long the logreceiver will wait for logsender to reply to commands. The default is 60; the legal range is 5 to 1000000.
 - v** Print version and exit.
- Additional arguments for SSL:
- C** *certFileName* Certificate in PEM format that will be sent to the peer upon request.
 - J** *certAuthFileName* Certificate authority (CA) file in PEM format to use for peer certificate verification.
 - K** *keyFileName* Private key in PEM format for the certificate (associated with the **-C** certificate).
 - Q** *passphrasestring* Private key passphrase. Required if the **-K** option is used and the private key is encrypted.

- S Enable SSL mode. Must be specified to use any other SSL options and must also be specified when starting the peer process.
- V Disable verification of the peer's certificate.

This utility starts up a logreceiver process. As part of a hot standby setup, the logreceiver process runs on the slave system to receive transaction log records, as they are generated, from a logsender process that is running on a master system.

The received transaction log records are written to files in a specified directory on the slave system. The slave system can run `continuousRestoreFromArchiveLogs`: to restore these transaction log records.

The logreceiver writes logging output to a file with the path and name or in the directory specified by the `-l` argument. If this file is a directory, the file name is `logreceiver_listeningPort.log`. If no `-l` argument is used, it writes to `/opt/gemstone/log/logreceiver_listeningPort.log`. If a file with the specified name exists, it is appended to. It is not deleted on process exit.

The logreceiver process will continue running until explicitly stopped using the **stoplogreceiver** command. If connection to the stone or the logsender process is lost, it will attempt to reconnect.

gslist will report running logreceiver processes.

To start the logsender using SSL to establish a secure connection between logsender and log receiver, both `startlogsender` and `startlogreceiver` can be provided with SSL credentials. When SSL arguments are provided, `startlogreceiver` will start the logreceiver in secure mode, requiring an SSL connection with the logsender. This is described under "Connecting using SSL Mode" on page 246

startlogsender

```
startlogsender -P listening port -A listeningAddress+ -T tranlogDir+ [ -I logFile ]
[ -s stoneName ] [ -u ] [ -d ] [ -v ] [ -h ] [ -C certFileName ] [ -J certAuthFileName ]
[ -K keyFileName ] [ -Q passphrasestring ] [ -S ] [ -V ]
```

- A *listeningAddress* The address (es) on which the logsender will listen for connections from logreceivers. At least one is required, up to 4 may be specified.
- d Print debug tracing of tranlog read and write operations to log file. The log file will be much larger.
- I *logFileOrDir* The path and filename or directory for the logged output of the logsender process. The default is `/opt/gemstone/log/logsender_listeningPort.log`.
- h Print usage and exit.
- P *listeningPort* The port or named service on which on which the logsender will listen for connections from logreceivers.
- s *stoneName* The name of the master stone. *stoneName* is required for the logsender to detect that stone has written new data to the master tranlogs. Without stone name, logsender will transmit tranlogs that it finds on the file system when it starts up.
- T *tranlogDir* Directory(s) where the master stone's transaction logs are located. Normally the same as stone's STN_TRAN_LOG_DIRECTORIES but may also include archive directories. logsender will examine these directories for new files to send. At least one is required, up to 20 -T may be specified.
- u Do not compress tranlog files at record level before transmission.
- v Print version and exit.

Additional arguments for SSL:

- C *certFileName* Certificate in PEM format that will be sent to the peer upon request.
- J *certAuthFileName* Certificate authority (CA) file in PEM format to use for peer certificate verification.
- K *keyFileName* Private key in PEM format for the certificate (-C option).
- Q *passphrasestring* Private key passphrase. Required if the -K option is used and the private key is encrypted.
- S Enable SSL mode. Must be specified to use any other SSL options and must also be specified when starting the peer process.
- V Disable verification of the peer's certificate.

This utility starts up a logsender process. As part of a hot standby setup, the logsender process runs on the master system to transmit transaction log records, as they are generated, to a logreceiver process that is running on a slave system.

The logsender writes logging output to a file with the path and name or in the directory specified by the `-l` argument. If this file is a directory, the file name is `logsender_listeningPort.log`. If no `-l` argument is used, it writes to `/opt/gemstone/log/logsender_listeningPort.log`. If a file with the specified name exists, it is appended to. It is not deleted on process exit.

The logsender process will continue running until explicitly stopped using the **stoplogsender** command. If connection to the stone is lost, it will attempt to reconnect.

gsl will report running logsender processes.

To start the logsender using SSL to establish a secure connection between logsender and log receiver, both `startlogsender` and `startlogreceiver` can be provided with SSL credentials. When SSL arguments are provided, `startlogsender` will start the logsender in secure mode, requiring an SSL connection with the logreceiver. This is described under “Connecting using SSL Mode” on page 246.

startnetldi

```
startnetldi [ netLdiName ] [ -g | [ -s ] [ -k keytab ] ] [ -a name ] [ -A addresses ]+ [ -b ] [ -d ]
           [ -D directoryPath ] [ -l logFile ] [ -n ] [ -P portNumber ] [ -r ] [ -t timeout ]
```

```
startnetldi -h | -v
```

<i>netLdiName</i>	The name or port number of the GemStone network server. If <i>netLdiName</i> is a numeric value equal to or less than 65535, it is interpreted as a port number. If the given port is in use, it will result in an error. Other values are interpreted as the netldi name. If the -P argument is omitted, this name is looked up in the network services database to determine the port number. Network resource syntax is not permitted. The default is <code>gs64ldi</code> .
-a <i>name</i>	Captive account; all child processes created by the NetLDI will belong to the account named <i>name</i> . By default, child processes belong to the client's account.
-A <i>addresses</i>	Address to listen on. Up to 10 arguments are accepted. If no -A arguments are provided, listening is on the default wildcard address <code>::</code> . If this default wildcard address is included, then other -A arguments are ignored. If the -A entry arguments do not include <code>::</code> or <code>::1</code> , then <code>::1</code> is also listened on.
-b	Maximum client connection backlog (default: 64).
-d	Debug mode; inserts more extensive messages in the log file.
-D <i>directoryPath</i>	Specifies a directory that will be used to compose log file paths for process log files for processes whose NRS includes the <code>%D</code> placeholder. When spawning a child process, the <code>%D</code> in the NRS is replaced with <i>directoryPath</i> before forking.
-g	Guest mode; no accesses are authenticated. This option is not allowed if the NetLDI's effective user id is the root account. Not compatible with -k .
-h	Print usage and exit.
-k	Enable Kerberos and set the keytab file name; logins with empty host password will authenticate using Kerberos. Not compatible with -g .
-l <i>logFile</i>	The logged output of the NetLDI; the default is <code>/opt/gemstone/log/NetLdiName.log</code>
-n	Do not allow any <i>ad hoc</i> processes to be created (ad hoc processes are ones not listed in <code>\$GEMSTONE/sys/services.dat</code>).
-N	Use numeric IP addresses for printing peer info in logs, do not do reverse DNS lookups to get hostnames of clients.
-P <i>portNumber</i>	The well-known port number that netldi will use.

- r** If there is a running NetLDI (version 3.3 or above) with this name, and that NetLDI has different version than specified in the current environment, stop the running NetLDI and restart it.
- s** Secure; require authentication for **all** NetLDI accesses.
- t *timeout*** Seconds to wait for a spawned client to start, such as a Gem; default is 30 seconds.
- v** Print version and exit

This command starts a GemStone network server with the specified *netLdiName* and *timeout* (given in seconds). The server spawns GemStone processes in response to login requests from remote applications and requests for remote repository access from Stone repository monitor processes. If your machine is slow or heavily loaded, and RPC logins time out before completing, specify a larger timeout value.

Legal characters in a NetLDI name are A-Z, a-z, 0-9, and the characters period, underscore, and dash (. _ -). NetLDI names with other characters are disallowed. NetLDI names should not start with period or dash, and should include at least one letter or digit. Also, NetLDI names must not match an existing system service, such as `ldap`, `syslog`, etc.

The NetLDI listens for requests, including RPC login requests, on a specified or configured port and optionally on a specific address. During the RPC login process, it uses a separate set of ports to establish communication between the Gem and its client. If you are running over a firewall, you can specify the port using the **-P** option, and configure your firewall to permit access via this port.

The locations and names of log files for process started by the NetLDI are defined by NRS directives in the login parameters or by settings in the `GEMSTONE_NRS_ALL`. Using the **-D *logfile*** argument to `startnetldi` sets a variable within the NetLDI, which can be optionally specified in the login parameters or `GEMSTONE_NRS_ALL` using the `%D` pattern.

NetLDI can be configured to authenticate starting processes, or all accesses. It may configure to authenticate the UNIX `userId` (including by using Kerberos) or to run in guest mode, which does not authenticate. These options work in conjunction with permissions for the extent files to provide security while also ensuring the appropriate access to GemStone for authorized users. How to configure the NetLDI is described in Chapter 4, “NetLDI and Interprocess Access”, starting on page 71.

For assistance with startup failures, refer to “To Troubleshoot NetLDI Startup Failures” on page 109.

netldid

`$GEMSTONE/sys/netldid` is the executable that is invoked by `startnetldi`. While `startnetldi` blocks until startup is complete, then returns control to the command line, executing `netldid` directly does not return. This can allow you to run the NetLDI as a background process.

The options described for `startnetldi` are also available for `netldid`, with the exception of **-r**.

startstone

startstone [*gemStoneName*] [**-l** *logFile*] [**-e** *exeConfig*] [**-z** *systemConfig*] [**-h** | **-v**] [**-R**] [**-N**]

<i>gemStoneName</i>	Name of the GemStone repository monitor, default is <code>gs64stone</code> . Network resource syntax is not permitted.
-C	startup for conversion; only applies when starting up a version that requires conversion. Refer to the Installation Guide for the specific version for details.
-e <i>exeConfig</i>	The GemStone executable configuration file. See “Executable Configuration File” on page 287.
-h	Print usage and exit.
-l <i>logFile</i>	The location (or filename) for the logged output of the stone; the default is (1) the setting of the <code>GEMSTONE_LOG</code> environment variable, if defined; (2) <code>\$GEMSTONE/data/gemStoneName.log</code>
-N	Do not replay transaction logs as part of startup. Unless used with the -R option, and transaction logs are replayed manually, work may be lost.
-R	Start up from the most recent checkpoint and go into restore mode. This allows transaction logs to be restored. Used when restoring from backup.
-v	Print version and exit
-z <i>systemConfig</i>	The GemStone system configuration file. See “System Configuration File” on page 286.

This command opens the GemStone repository specified by the configuration files. For details on how **startstone** determines which the configuration file/s to use, see “How GemStone Uses Configuration Files” on page 285.)

Legal characters in a Stone name are A-Z, a-z, 0-9, and the characters period, underscore, and dash (`._-`). Stone names with other characters are disallowed. Stone names should not start with period or dash, and should include at least one letter or digit. Also, Stone names must not match an existing system service, such as `ldap`, `syslog`, etc.

The **-N** option is intended for use when the repository needs recovery, but the transaction logs specified in the configuration file cannot be found or have become corrupted. The **-N** forces the repository to start up anyway, even though transactions committed since the last checkpoint will be lost. A new transaction log will be initialized as part of the startup.

The **-R** option is used when restoring from backup. This starts up the repository at the point of the most recent checkpoint in the extents, and puts the repository into restore mode. You can then invoke Topaz to restore from transaction logs and commit the restored state.

If the extents against which Stone is being started require recovery, or if you are restoring from online extent backups, then you must specify the **-N** option along with the **-R** option; otherwise, an error will result and the repository monitor will not start.

For startup failures, refer to “To Troubleshoot Stone Startup Failures” on page 104.

stoned

`$GEMSTONE/sys/stoned` is the executable that is invoked by **startstone**. While **startstone** blocks until startup is complete, then returns control to the command line, executing **stoned** directly does not return. This can allow you to run the Stone as a background process.

The options described for **startstone** are also available for **stoned**.

statmonitor

statmonitor *stoneName* [*options, see below*]

<i>stoneName</i>	Required; the name of the Stone to monitor.
-A	Collect all available system statistics.
-B count	Number of persistent shared counters to sample. Maximum is 1536, default is 0.
-C	Collect system statistics for each individual CPU.
-d directory	Directory where output files are written. Cannot be used with the -f or -F options; to specify a directory with those options, include it in the filename or pattern.
-D	Collect system statistics for all disks and partitions.
-f fileName	The output file name. By default, the output filename is <code>statmonN.out</code> , where <i>N</i> is the process ID. If this file already exists, <code>statmonitor</code> will append an <i>-N</i> to the filename, where <i>N</i> starts at 1 and increments as needed. To send output to stdout instead of a file, specify -f stdout . May not be used with the -d option.
-F pattern	Pattern used to generate the file name. Values in the pattern starting with a single '%' character have the meaning described in the <code>strftime(3)</code> function call. Additionally, the following patterns which start with double '%' characters are also accepted: %%C - name of the shared page cache %%H - name of the host %%i - sample interval in seconds %%I - sample interval in milliseconds %%P - process ID of <code>statmonitor</code> %%S - name of the stone May not be used with the -d option.
-h	With no argument, print usage and exit.
-h hours	The number of hours (default: forever).
-i interval	The interval in seconds (default: 20). Select either -i or -I .
-I intervalMs	The interval in milliseconds (default 20000; minimum 10). Select either -i or -I .
-J	Sample the Stone, shared cache monitor, and page manager only.

-k <i>listOfTimes</i>	List of times at which statmonitor should be restarted. The restart is performed at the next sample interval after the specified time has passed. <ul style="list-style-type: none"> ▶ Either the -r or -R command must also be specified. ▶ List must start and end with a single quote. ▶ times include hours and minutes, colon separated. ▶ hours are in 24-hour format. ▶ Multiple times are comma separated. ▶ Duplicates should not be included.
-n <i>numAppStats</i>	The number of application statistics (default: 0).
-N	Collect system statistics for all network interfaces.
-p <i>sessionId</i>	A GemStone sessionId to monitor. You may specify multiple sessions. (Default: monitor all sessions.)
-P	Sample the Stone, shared cache monitor and AIO page server threads only.
-q	Quiet mode. Only print messages if an error occurs.
-Q <i>pid1,pid2,...</i>	Record statistics for a list of process IDs.
-r	Restart a new output file when the current one completes. Each file will be given a unique name. This option may only be used with the -h or -t switches, which control when a restart is done.
-R	Same as -r except the output file name is regenerated for each restart if the -F option is used.
-S	Sample only the Stone and shared cache monitor.
-t <i>times</i>	The maximum number of samples to collect before starting a new output file (default is forever). Select either -h or -t .
-u <i>seconds</i>	The maximum number of seconds to wait before flushing the cached information to the output file (default: 60). If 0 then the flush will be done every interval.
-U <i>uid1,uid2,...</i>	Record statistics for all processes owned by one or more UNIX user IDs.
-v	Print version and exit
-W	Collect system statistics for system memory pages (Solaris only).
-X	Run in host-only mode. Sample host system statistics only and do not attach to a shared page cache. May be combined with other flags EXCEPT: -m , -n , -p , -P , -S , or -Y .
-Y	Disable collection of all system statistics, including per-process data.

- z Write the output in compressed gzip format.
- Z Write the output in compressed lz4 format.

Record statistics for a repository and/or the operating system to a disk file. **statmonitor** runs in the background, sampling specified data at a specified interval and recording the data to a text file. The data in this file can be viewed by the graphical application VSD. See **vsd** (page 366) and the *VSD Users Guide* for more information.

Statistics are collected from the shared page cache. Only data for GemStone processes attached to the shared page cache on the host on which statmonitor is running are collected. To monitor Gems on systems with remote Gem servers, you must run statmonitor both on the Stone's machine and on the Gem server machine.

The *stonename* argument is used to specify the cache to monitor, both for caches that are local to the Stone and caches that are remote from the Stone with that name. Configurations in which a single machine is hosting remote caches for multiple stones that are running with the same Stone name are ambiguous and will not work correctly; this configuration is strongly discouraged.

In addition to running statmonitor from the command line, you can also specify that statmonitor be automatically started at stone startup, and the startup for remote gem caches and remote mid-level caches, to ensure statmonitor data is collected. See the configuration parameters STN_STATMONITOR_ARGS (page 328), GEM_STATMONITOR_ARGS (page 303), and GEM_STATMONITOR_MID_CACHE_ARGS (page 303).

stoplogreceiver

stoplogreceiver **-P** *listeningPort* [**-v**] [**-h**]

- h** Print usage and exit.
- P** *listeningPort* The port that the logreceiver is listening on for **stoplogreceiver** connections. If **-p** was used in the **startlogreceiver** command, that port must be specified here; otherwise the port specified by the **-P** argument to **startlogreceiver**.
- v** Print version and exit.

This utility stops a logreceiver process that was started by a previous **startlogreceiver** command. This may only be executed on the same node as the logreceiver is running.

stoplogsender

stoplogsender **-P** *listeningPort* [**-v**] [**-h**]

-h Print usage and exit.

-P *listeningPort* The port on which this logsender is listening. It must be the same port as specified in the **-P** argument of the **startlogsender**.

-v Print version and exit

This utility stops a logsender process that was started by a previous **startlogsender** command. This may only be executed on the same node as the logsender is running.

stopnetldi

stopnetldi [*netLdiName*] [**-h**] [**-v**]

netLdiName The name of the GemStone network server; the default is gs64ldi.
Network resource syntax is not permitted.

-h Print usage and exit.

-v Print version and exit

Gracefully stop a GemStone network server.

If *name* is not specified, the name of the NetLDI to stop is taken from GEMSTONE_NRS_ALL environment variable, if defined there. If GEMSTONE_NRS_ALL does not define a NetLDI name, then a NetLDI with the default name gs64ldi is stopped.

If a portNumber was used to start the netldi process, then the portNumber can be used to stop the netldi process.

stopstone

stopstone [*gemStoneName* [*gemStoneUserName* [*gemStonePassword*]]] [**-i**] [**-t**] [**-h**]

<i>gemStoneName</i>	The name of the GemStone repository monitor, commonly <code>gs64stone</code> . Network resource syntax is not permitted.
<i>gemStoneUserName</i>	A privileged GemStone user account name, such as "DataCurator" or "SystemUser".
<i>gemStonePassword</i>	The GemStone password for the specified <i>gemStoneUserName</i> .
-h	Print usage and exit.
-i	Causes any current GemStone sessions to be terminated immediately.
-t	Specifies how long to wait for other processes to detach from the cache. Default is -1, wait forever.

Gracefully shut down a GemStone repository monitor. In the process, a checkpoint is performed in which all committed transactions are written to the extents. If you specify the **-i** (immediate) option, the repository monitor is shut down even if there are GemStone users logged in. The **-t** option specifies how long to wait for all session to detach from the cache before returning; if omitted, **stopstone** will wait forever. If you do not supply the *gemStoneName*, *gemStoneUserName*, and *gemStonePassword* on the command line, this command will prompt you for them.

Because this command uses a Gem session process to connect to *gemStoneName*, it fails if the Gem is unable to connect for any reason, such as inability to attach a shared page cache. For assistance, refer to "To Troubleshoot Session Login Failures" on page 113.

topaz

topaz [-h | -v]

topaz [-r] [-q] [-i | -I *topazini*] [-u *useName*] [-- *other args*]

topaz -l | -L [-q] [-i | -I *topazini*] [-u *useName*] [-e *exeConfig*] [-z *systemConfig*]
[-T *tocSizeKB*] [-C *configParams*] [-- *other args*]

- C** *configParams* Provides configuration parameters, in configuration file syntax, that override settings in the configuration files. Only applies to linked sessions (RPC sessions may use the -C syntax in the Gem's NRS).
- e** *exeConfig* The GemStone executable configuration file (only allowed with linked sessions). See "Executable Configuration File" on page 287.
- h** Print usage and exit.
- i** Ignore the initialization file, *.topazini*.
- I** *topazini* Specify a complete path and file to a topazini initialization files, and use this rather than any *.topazini* in the default location.
- l** Invoke the linked version of Topaz.
- L** Invoke the linked version of Topaz, and do not apply any command **set gemnetid** that may appear in the *.topazini* file or a file passed in using **-I**.
- q** Start Topaz in quiet mode, suppressing printout of the banner and other information.
- r** Invoke the RPC (remote procedure call) version of Topaz.
- T** *tocSizeKB* The GEM_TEMPOBJ_CACHE_SIZE that will be used. Overrides any settings provided in configuration files passed as arguments with the **-e** or **-z** options. Only applies to linked sessions.
- u** *useName* Sets the cache name, as recorded by statmonitor for viewing in VSD. This is also useful for identifying processes in OS utilities such as *top* or *ps*.
- v** Print version and exit.
- w** On Windows client only; forces terminal behavior regardless of I/O device.
- z** *systemConfig* The GemStone system configuration file (only allowed with linked sessions). See "System Configuration File" on page 286.
- *otherArgs* Arbitrary text arguments *otherArgs* may be included after the "--" end of arguments marker, which must follow any of the above topaz arguments are included.

This command invokes various forms of topaz. The default is to invoke the remote procedure call (RPC) version of topaz; to do this explicitly, use the -r option.

To invoke the linked version of topaz, which is recommended or required for some maintenance operations, use the `-l` or `-L` option. The advantage of the `-L` option is that `.topaz.ini` initialization file command cannot inadvertently configure the login to be RPC rather than linked.

Several topaz arguments only apply in linked topaz. The arguments `-exeConfig` and `-systemConfig` determine the configuration files that linked topaz reads. For more information about this, see “How GemStone Uses Configuration Files” on page 285.

Settings within topaz can allow linked topaz to perform RPC logins.

Topaz is available for Windows, in addition to regular server platforms, as part of the Windows Client installation. Options are more limited since linked logins are not possible.

For further information, see the *Topaz Programming Environment*.

vsd

vsd [**-b** *color*] [**-u**] [[**-a**] *statmonDataFile+*]

vsd -h | -v

-a Append mode. Files following the **-a** are appended to the first one; the data in these files is merged so statistics are viewed as if they were a single file.

-b *color* Set the master background color. Color may be a hex RGB value (e.g.: #d3d3ff) or a valid color name (e.g.: white). The list of recognized color names may be found at:
<http://www.tcl.tk/man/tcl8.6/TkCmd/colors.htm>

-h Displays a usage line and exits.

-v Print version information and exit.

statmonDataFile The name of a statistics data file to load into the VSD as it is started.

This command starts VSD, the graphical tool to view and analyze statmonitor data. The default is to open an empty instance of VSD, into which you can then load one or multiple statmonitor data files. After loading data files, you may select one or multiple GemStone processes, and view charts on one or multiple statistics for these processes.

VSD is available for Windows, in addition to regular server platforms, as part of the Windows Client installation.

For a complete description of VSD, see the *VSD User's Guide*.

waitstone

waitstone [**-h**] [*gemStoneName* | *netLdiName*] [*timeout*] [*waitForWarming*]

-h	Display a usage line and exits
<i>gemStoneName</i>	The name of the GemStone repository monitor.
<i>netLdiName</i>	The name of the GemStone netldi service.
<i>timeout</i>	How many seconds to wait for GemStone to initialize before reporting a problem. The default (0) means wait forever; -1 means don't wait, try once and return the result. Only valid when specifying either <i>gemStoneName</i> or <i>netLdiName</i> .
<i>waitForWarming</i>	If larger than 0, block until cache warmers finish.

This command reports whether the GemStone repository *gemStoneName* is ready to accept logins or whether *netLdiName* is ready to accept requests. If neither *gemStoneName* nor *netLdiName* are specified, **waitstone** will report on the default stone name, *gs64stone*.

Waitstone checks every 0.5 seconds to see if the Stone or NetLDI is ready. When the service is ready, **waitstone** issues a message to `stdout`. If the service is not ready by the time the specified number of seconds have elapsed, **waitstone** reports an error.

With a -1 timeout, **waitstone** will make one connection attempt to the Stone or NetLDI. Note that if the stone is inaccessible, the timeout may be longer, since the configured TCP/IP timeout may apply. This can result in a 20 second timeout regardless of the specified *timeout* value.

Use of *timeout* requires that the Stone or NetLDI name is also specified.

If a positive value for the *waitForWarming* argument is specified, **waitstone** will block until cache warming completes; this is useful when cache warming is automatically run on startup. The specific value of the *waitForWarming* is not important. Use of the *waitForWarming* positional argument requires specifying each of the preceding arguments.

You may specify the *gemStoneName* and *netLdiName* arguments as a GemStone network resource string. (See Appendix C, "Network Resource Strings (NRS)")

This command returns 0 exit status if the operation is successful; otherwise, it returns a non-zero value.

Network Resource Strings (NRS)

The network resource string (NRS) is a GemStone-specific way to format a string that contains:

- ▶ unique identifiers for a GemStone file or process within a networked environment, such as the node and name of a running Stone.
- ▶ specific instructions for performing actions, such as spawning a Gem process.
- ▶ configuration details and arguments, such as authentication information or locations for writing log files.

A basic understanding of NRS to compose login parameters is unavoidable, for any but the most simple configurations. However, NRS has a number of features and syntactical elements and a full understanding is not needed.

A.1 Using NRS

The primary use for NRS by applications and end users are in login parameters, to specify the Stone, and for RPC logins, to specify the Gem service. Some utility commands also accept NRS in order to perform operations remotely.

The following table provides the most common formats for NRS syntax for Stone and Gem services. In these examples, you may use the IP address instead of the hostname, and the NetLDI listening port instead of the NetLDI name.

Table A.1 Examples for common NRS formats

Stone or Gem Service	NRS
Local stone named <code>devstone</code>	<code>devstone</code>
Gem service specifying the gem on local node, where NetLDI is named <code>gs64ldi</code>	<code>gemnetobject</code>

Table A.1 Examples for common NRS formats

Stone named devstone, running on node oboe. This example applies when oboe is remote or local. If remote, then there must be a NetLDI named gs64ldi running on oboe.	!@oboe!devstone
Gem service specifying the gem to run on node oboe, where oboe is running NetLDI named gs64ldi.	!@oboe!gemnetobject
Gem service specifying the Gem to run on the local node, with local NetLDI named devldi.	!#netldi:devldi!gemnetobject
Specify a Stone named devstone, running on node oboe, with the NetLDI on oboe named devldi.	!@oboe#netldi:devldi!devstone
Gem service specifying the Gem to run on node oboe, with the NetLDI on oboe named devldi.	!@oboe#netldi:devldi!gemnetobject

GsNetworkResourceString

This GemStone Smalltalk class provides a programmatic API to construct correctly formatted NRS strings. The resulting string is in the most complete form, including default information, not in the most simplified form that you can enter as a parameter.

For example, the following expression recreates the NRS that can be used for the gemnetid parameter, based on the current RPC session.

```
topaz 2> run
GsNetworkResourceString defaultGemNRSFromCurrent asString
%
!tcp@santiam#netldi:52331#task!gemnetobject
```

See the image for more information.

GEMSTONE_NRS_ALL

The environment variable GEMSTONE_NRS_ALL determines which modifiers GemStone will use by default in each NRS it processes on your behalf.

This is frequently used to configure a specific NetLDI name when the default NetLDI name is not used, avoiding having to specify the NetLDI name in every NRS. For example,

```
$ GEMSTONE_NRS_ALL="#netldi:devldi"
$ export GEMSTONE_NRS_ALL
```

GEMSTONE_NRS_ALL can handle multiple and more complicated instructions, such as causing all Gem session process logs to be created with a specific name in a specific directory.

As an environment variable, it only affects processes created after it is set, and changing it may require restarting the NetLDI.

- ▶ If you set `GEMSTONE_NRS_ALL` before starting a NetLDI, which is a system-wide service, that setting is passed to all its children and becomes the default for all users of that service, i.e. RPC gems.
- ▶ If you set `GEMSTONE_NRS_ALL` before starting a Stone, an application, or a utility (such as `copydbf`), that setting applies only to your own processes and does not affect other users.

Because these settings are defaults, they take effect only if an explicit setting is not provided for the same modifier in a specific NRS setting or command-line argument.

For example, by default for an RPC login that specifies the default gemnetobject to spawn a gem, the gem log is written to the user's home directory. To use the `#dir` modifier to set the current (working) directory for NetLDI child processes, use code such as the following:

For example:

```
$ GEMSTONE_NRS_ALL=#dir:/user2/apps/logs
$ export GEMSTONE_NRS_ALL
```

This causes the gemnetobject log files to be written to the directory `/user2/apps/logs`; if this directory is not writable, an error is generated.

However, if the RPC login parameters specify a different setting for the name attribute, for example:

```
topaz > set gemnetid !#dir:/someotherdir!gemnetobject
```

This will be used, not the setting in `GEMSTONE_NRS_ALL`

If the client GCI application (e.g. `topaz`) has a `GEMSTONE_NRS_ALL` set to a value that is different from the NetLDI's `GEMSTONE_NRS_ALL`, the client application's directives takes precedence.

Controlling log file names and locations

The NRS in the Gem service login parameters, or the NRS specified by a `GEMSTONE_NRS_ALL` statement, can be used to configure the specific directory location and log file name for Gem logs and the logs for server process on remote nodes.

The `#dir` and `#log` directives in the NRS can combine the desired directory or file names with combinations of the following patterns, to ensure each log file is unique and identifiable:

```
%H - home directory
%D - the value of any -D option specified for the NetLDI
%M - machine's network node name
%N - executable's base name, such as gemnetid for a Gem
%P - process pid
%U - user name
```

A `#dir` directive must specify an existing, writable directory and end with a `/` path divider. The `#log` directive should not be a directory.

For example, a Gem with pid 27522 running on node `santiam` has by default a log file named `gemnetobject27522santiam.log` in the user's home directory. This could be

specified in an NRS using `#log:%H%N%P%M.log`. The syntax to specify this in a topaz login parameters would be:

```
set gemnetid !#log:%H%N%P%M.log!gemnetobject
```

To avoid the need to specify this in the login parameters for each login, it can be specified in a setting for the environment variable `GEMSTONE_NRS_ALL` that is set in the environment for all users.

```
setenv GEMSTONE_NRS_ALL "#log:%H%N%P%M.log"
```

“D” Pattern

Another level of control for log file locations is managed by the NetLDI. When the `-D` option to `startnetldi` is used, it sets a variable within the NetLDI that can be later accessed by using the `%D` pattern in the login parameter NRS for log or dir. This allows the individual Gem login parameters to remain the same, while the actual directories used can be changed at once for all users.

For example, the following respective command and environment variable settings would create Gem log files with the default name in the directory `/node1/users/gadmin/sales/logs/`.

Command line to start netldi:

```
startnetldi -D /node1/users/gadmin/sales/logs/
```

Gem NRS:

```
set gemnetid !#netldi:salesldi#log:%D/%N%P%M.log!gemnetobject
```

alternatively, the Gem NRS can be specified in the environment of all users:

```
setenv GEMSTONE_NRS_ALL "#netldi:salesldi#log:%D/%N%P%M.log"
```

If the desired location for log file changes, the only action required is to restart the NetLDI with the new directory. If your users log in to multiple repositories and you wish to keep the log files separate, this is easily done by using two NetLDIs with the appropriate `-D` arguments.

However, note that this way of managing the log file locations requires some attention:

- ▶ All Gems for which the NetLDI `-D` directory should be used will need to use the correct NRS. Any logins that do not use the `%D` within the login parameters will write their log files to the default locations.
- ▶ Care must be taken in making sure the `-D` is consistently used with the NetLDI, so the combination of the pattern in the NRS and the directory in the NetLDI will always produce a valid, writable path and filename; invalid log file names cause the login to fail.

If the `%D` is present in the NRS, and the NetLDI was not started with a `-D` argument, an empty string is used for `%D`. This means, for example, that if the `-D` argument is set to an absolute path and the `%D` in the NRS is used accordingly, then if NetLDI happens to get restarted without the `-D` argument, logs may attempt to get written to the root directory.

A.2 NRS Syntax

The BNF below provides full NRS syntax. Many of the options are used internally for interprocess communication, and are not intended for general use.

These strings may appear in the full form in places where command arguments are recorded, such as GemStone log files.

Spaces and Special Characters

An NRS can contain spaces and special characters; in particular, `!`, `#`, and `@` are frequently used.

While GemStone internally ensures that space and special characters are handled correctly, you may need to escape certain characters in NRS strings that are entered at a command prompt. The specific characters and escapes may vary on different UNIX shells.

```
% copydbf $GEMSTONE/data/extent0.dbf
  \!@node#auth:username@password#dbf\!/users/extent0.dbf_copy
```

If there is a space in the NRS, you can replace the space with a colon (`:`), or you can enclose the string in quotes (`" "`).

For example, the following network resource strings are equivalent:

```
% waitstone !@oboe#auth:user@password!gs64stone
% waitstone "!@oboe#auth user@password!gs64stone"
```

Syntax

nrs ::= [*nrs-header*] *nrs-body*

where:

nrs-header ::= ! [*address-modifier*] {*keyword-modifier*} [*resource-modifier*]!

All modifiers are optional, and defaults apply if a modifier is omitted. The value of an environment variable can be placed in an NRS by preceding the name of the variable with `"$"`. If the name needs to be followed by alphanumeric text, then it can be bracketed by `"{"` and `"}"`. If an environment variable named `foo` exists, then either of the following will cause it to be expanded: `$foo` or `${foo}`. Environment variables are only expanded in the *nrs-header*. The *nrs-body* is never parsed.

address-modifier ::= [**tcp**] [*@ node*]

Specifies where the network resource is.

node ::= *nrs-identifier*

If no node is specified, the current machine's network node name is used. The identifier may also be an Internet-style numeric address. For example:

```
!@120.0.0.4#server!cornerstone
```

nrs-identifier ::= *identifier*

Identifiers are runs of characters; the special characters `!`, `#`, `$`, `@`, `^` and white space (blank, tab, newline) must be preceded by a `"^"`. Identifiers are words in the UNIX sense.

keyword-modifier ::= (*authorization-modifier* | *environment-modifier*)

Keyword modifiers may be given in any order. If a keyword modifier is specified more

than once, the latter replaces the former. If a keyword modifier takes an argument, then the keyword may be separated from the argument by a space or a colon.

authorization-modifier ::= ((#auth | #encrypted) [:] username [@ password])

#auth specifies a valid OS user name on the target network. A valid OS user password is needed only if the resource type requires authentication. **#encrypted** is used by GemStone utilities. This type of authorization is the default.

username ::= nrs-identifier

If no OS user name is specified, the default is the current OS user. See the earlier discussion of *authorization-modifier*.

password ::= nrs-identifier

Only needed if the resource type requires authentication; see the earlier discussion of *authorization-modifier*.

environment-modifier ::= (#netldi | #dir | #log) [:] nrs-identifier

#netldi causes the named NetLDI to be used to service the request. If no NetLDI is specified, the default is `gs64ldi`. When you specify the **#netldi** option, the *nrs-identifier* is either the name of a NetLDI service or the port number at which a NetLDI is running.

#dir sets the default directory of the network resource. It has no effect if the resource already exists. If a directory is not set, the pattern “%H” (home directory) is used. (See the definition of *nrs-identifier*.)

#log sets the name of the log file of the network resource. It has no effect if the resource already exists. If the log name is a relative path, it is relative to the working directory. If a log name is not set, the pattern “%N%P%M.log” is used, as described below, and following the syntax in the definition of *nrs-identifier*

The argument to **#dir** or **#log** can contain patterns that are expanded in the context of the created resource. The following patterns are supported:

%H	home directory
%D	a directory path supplied by the NetLDI
%M	machine’s network node name
%N	executable’s base name, such as <code>gemnetid</code> for a Gem
%P	process pid
%U	user name
%%	(escaped) %

resource-modifier ::= (#server | #spawn | #task | #dbf | #monitor | #file)

Identifies the intended purpose of the string in the *nrs-body*. An NRS can contain only one resource modifier. The default resource modifier is context sensitive. For instance, if the system expects an NRS for a database file, then the default is **#dbf**.

#server directs the NetLDI to search for the network address of a server, such as a Stone or another NetLDI. If successful, it returns the address. The *nrs-body* is a network server name. A successful lookup means only that the service has been defined; it does not indicate whether the service is currently running. A new process will not be started. (Authorization is needed only if the NetLDI is on a remote node and is running in secure mode.)

#task starts a new Gem. The *nrs-body* is a NetLDI service name (such as “gemnetobject”), followed by arguments to the command line. The NetLDI creates the named service by looking first for an entry in `$GEMSTONE/sys/services.dat`, and

then in the user's home directory for an executable having that name. The NetLDI returns the network address of the service. (Authorization is needed to create a new process unless the NetLDI is in guest mode.) The **#task** resource modifier is also used internally to create page servers.

#dbf is used to access a database file. The *nrs-body* is the file spec of a GemStone database file. The NetLDI creates a page server on the given node to access the database and returns the network address of the page server. (Authorization is needed unless the NetLDI is in guest mode).

#spawn is used internally to start the garbage collection and other service Gem processes.

#monitor is used internally to start up a shared page cache monitor.

#file means the *nrs-body* is the file spec of a file on the given host (not currently implemented).

nrs-body ::= unformatted text, to end of string

The *nrs-body* is interpreted according to the context established by the *resource-modifier*. No extended identifier expansion is done in the *nrs-body*, and no special escapes are needed.

GemStone Kernel Objects

This appendix describes the predefined objects that are located in a freshly installed GemStone/S 64 Bit repository.

Non-Numeric Constants

The following non-numeric constants are defined in the Globals dictionary and protected by the SystemObjectSecurityPolicy:

- ▶ **true** (an instance of Boolean)
- ▶ **false** (an instance of Boolean)
- ▶ **nil** (an instance of UndefinedObject)

Numeric Constants

Floating point constants are instances of class Float or class DecimalFloat. They are defined in the Globals dictionary and are protected by the SystemObjectSecurityPolicy. Refer to the *Programming Guide* for more information on these values.

DecimalPlusInfinity
DecimalMinusInfinity
DecimalPlusQuietNaN
DecimalMinusQuietNaN
DecimalPlusSignalingNaN
DecimalMinusSignalingNaN
PlusInfinity
MinusInfinity
PlusQuietNaN
MinusQuietNaN
PlusSignalingNaN
MinusSignalingNaN

Repository and GsObjectSecurityPolicies

The Repository is the root of a GemStone system. This structure contains the GsObjectSecurityPolicies, which implement object-level security. For more information, refer to the *Programming Guide* for more information on the meaning and use of these objects.

SystemRepository. This single instance of Repository is defined in the Globals dictionary. Repository is a subclass of Collection, and the indexable part of SystemRepository contains references to all the security policies (all the instances of GsObjectSecurityPolicy) in GemStone.

The SystemRepository object initially contains eight security policies, three of which are public and named: the SystemObjectSecurityPolicy (owned by the SystemUser), the DataCuratorObjectSecurityPolicy (owned by the DataCurator), and the PublishedObjectSecurityPolicy (owned by SystemUser). The SystemRepository object itself is protected by the DataCuratorObjectSecurityPolicy. New GsObjectSecurityPolicies may be created and added to the SystemRepository object, using the methods `new`, `newInRepository:`, or by some methods that create new users.

For more on GsObjectSecurityPolicies, see the *Programming Guide for GemStone/S 64 Bit*.

SystemObjectSecurityPolicy. This security policy is defined in the Globals dictionary. For backwards compatibility, the key `#SystemSegment` also refers to this security policy, and may be used in upgraded repositories. The SystemObjectSecurityPolicy object itself is protected by the DataCuratorObjectSecurityPolicy.

The SystemObjectSecurityPolicy is the default security policy for its owner, the SystemUser (who has write authorization for any of the objects in this security policy). The “world” (that is, the set of all GemStone users) is authorized to read, but not write, the objects in this security policy. In addition, the group `#System` is authorized to write in this security policy.

DataCuratorObjectSecurityPolicy. This security policy is defined in the Globals dictionary. For backwards compatibility, the key `#DataCuratorSegment` also refers to this security policy, and may be used in upgraded repositories. The DataCuratorObjectSecurityPolicy object itself is protected by the DataCuratorObjectSecurityPolicy.

The DataCuratorObjectSecurityPolicy is the default security policy for its owner, the DataCurator (who has write authorization for any of the objects in this security policy). The “world” (that is, the set of all GemStone users) is authorized to read, but not write, the objects in this security policy. In addition, the `#DataCuratorGroup` is authorized to write in this security policy.

Objects in the DataCuratorObjectSecurityPolicy include the Globals dictionary, the SystemRepository object, most instances of GsObjectSecurityPolicy, AllUsers (the set of all GemStone UserProfiles), AllGroups (the collection of groups authorized to read and write objects in security policies), and each UserProfile object.

PublishedObjectSecurityPolicy. This security policy is defined in the Globals dictionary. For backwards compatibility, the key `#PublishedSegment` also refers to this security policy, and may be used in upgraded repositories. The PublishedObjectSecurityPolicy object itself is protected by the DataCuratorObjectSecurityPolicy.

The `PublishedObjectSecurityPolicy` is owned by the `SystemUser`. The group `#Subscribers` is authorized to read in this security policy. The group `#Publishers` is authorized to read and write in this security policy. The “world” is not authorized to read or write the objects in this security policy.

Global Variables and Collections

AllUsers. The `AllUsers` object is an instance of `UserProfileSet`. It is defined in `Globals`, and is protected by the `DataCuratorObjectSecurityPolicy`. `AllUsers` contains the `UserProfiles` of all GemStone users. When GemStone is first installed, `AllUsers` contains five `UserProfiles`: `SystemUser`, `DataCurator`, `GcUser`, `SymbolUser`, and `Nameless`. For more information on the Special system accounts, see “Special System Users” on page 146.

AllDeletedUsers. This collection is defined in the `Globals` dictionary, and is protected by the `DataCuratorObjectSecurityPolicy`. `AllDeletedUsers` contains `DeletedUserProfiles` representing GemStone users that have been removed from `AllUsers`. When GemStone is first installed, `AllDeletedUsers` is empty.

AllGroups. This dictionary is defined in `Globals`, and is protected by the `DataCuratorObjectSecurityPolicy`. Each `Symbol` in `AllGroups` corresponds to an instance of `UserProfileGroup`. When GemStone is first installed, `AllGroups` contains `UserProfileGroups` named `System`, `Publishers`, `Subscribers`, `DataCuratorGroup`, and `SymbolUser`.

AllKerberosPrincipals. This dictionary is defined in `Globals`, and is protected by the `DataCuratorObjectSecurityPolicy`. Each `Symbol` in `AllKerberosPrincipals` corresponds to an instance of `KerberosPrincipal`. This dictionary is empty unless `KerberosPrincipals` have been explicitly created.

AllClusterBuckets. This `ClusterBucketArray` is defined in the `Globals` dictionary, and is protected by the `DataCuratorObjectSecurityPolicy`. `AllClusterBuckets` contains instances of `ClusterBucket`, which group objects on extent pages to improve performance. When GemStone is first installed, `AllClusterBuckets` contains the following predefined cluster buckets (listed by cluster id):

1. A generic bucket whose extent is “don’t care”. This bucket, the current default after session login, is invariant and may not be modified.
2. A generic bucket whose extent is “don’t care”.
3. A generic bucket whose extent is “don’t care”.
4. The kernel classes “behaviorBucket”, extent 1.
5. The kernel classes “descriptionBucket”, extent 1.
6. The kernel classes “otherBucket”, extent 1.
7. A generic bucket whose extent is “don’t care”.

ConfigurationParameterDict. This dictionary is defined in the `Globals` dictionary, and is protected by the `SystemObjectSecurityPolicy`. Its keys list the names of the configuration parameters available to a session. Its values are only used internally in GemStone, to locate the values of the parameters themselves for an individual session.

DbfHistory. This String describes the history of this repository, from the version in which it was first created, and each subsequent upgrade.

DbfOrigin. This SmallInteger identifies the version in which this repository was first created, if the repository originated in v3.2 or later.

DeprecationEnabled. This is nil, or a keyword. When deprecation is enabled, it will be set to the configured handling instructions when a deprecated method is encountered. Deprecation is described in the *Programming Guide*.

ErrorSymbols. This SymbolDictionary is defined in the Globals dictionary, and is protected by the SystemObjectSecurityPolicy. It maps mnemonic symbols to error numbers.

GciStructsMd5String and **GciTsStructsMd5String.** These constants encode the GCI structures that this version of GemStone uses. On upgrade, they can be compared to determine if there are changes in the GCI structures that would impact C code using GemBuilder for C.

GemStoneError. This SymbolDictionary is defined in the Globals dictionary, and is protected by the DataCuratorObjectSecurityPolicy. Each key is a Symbol representing a native language, and is associated with an Array of error messages in that language. Initially, this dictionary contains the single key #English.

GemStone_Legacy_Streams. This SymbolDictionary contains classes implementing the legacy GemStone PositionableStream interface. See the *Programming Guide* for more information.

GemStone_Portable_Streams. This SymbolDictionary contains classes implementing the PositionableStream interface that is ANSI-complaint and portable to other Smalltalk dialects. See the *Programming Guide* for more information.

InstancesDisallowed. This IdentitySet is defined in the Globals dictionary, and is protected by the SystemObjectSecurityPolicy. This collection is used for error reporting for some cases where instance creation is disallowed.

NativeLanguage. This Symbol is not used in this version, but may exist in upgraded repositories.

NotTranloggedGlobals. This SymbolDictionary is defined in the Globals dictionary, and is protected by the DataCuratorObjectSecurityPolicy. This collection holds objects for which changes are committed but not recorded in the transaction logs. When GemStone is first installed, NotTranloggedGlobals is empty. NotTranloggedGlobals is described in the *Programming Guide*.

GsDefaultIndexOptions. This instance of GsIndexOptions provides the default indexing internal structures.

StringConfiguration. This is the default class used to instantiate literal strings, and to control the behavior of String comparison, including String equality, between traditional and Unicode strings. By default, this is set to the String class. When it is set to Unicode16 class, the repository is in Unicode Comparison Mode. Changing the value should be done with great caution, since it may break existing collections of Strings and other usages that depend on string comparison. Unicode Comparison Mode is described in the *Programming Guide*.

Transcript. An instance of TranscriptStreamPortable.

Table A.1 Initial Contents of the Globals Dictionary

	Key	The object's class
Numeric Constants	#DecimalMinusInfinity	DecimalFloat
	#DecimalMinusQuietNaN	DecimalFloat
	#DecimalMinusSignalingNaN	DecimalFloat
	#DecimalPlusInfinity	DecimalFloat
	#DecimalPlusQuietNaN	DecimalFloat
	#DecimalPlusSignalingNaN	DecimalFloat
	#MinusInfinity	Float
	#MinusQuietNaN	Float
	#MinusSignalingNaN	Float
	#PlusInfinity	Float
	#PlusQuietNaN	Float
	#PlusSignalingNaN	Float
	Non-Numeric Constants	#false
#nil		UndefinedObject
#true		Boolean
Repository and instances of GsObjectSecurityPolicy	#DataCuratorObjectSecurityPolicy, #DataCuratorSegment	GsObjectSecurityPolicy
	#DbfHistory	String
	#DbfOrigin	SmallInteger
	#PositionableStream_position	String
	#PublishedObjectSecurityPolicy, #PublishedSegment	GsObjectSecurityPolicy
	#SystemRepository	Repository
	#SystemObjectSecurityPolicy, #SystemSegment	GsObjectSecurityPolicy

Table A.1 Initial Contents of the Globals Dictionary (Continued)

	Key	The object's class
Collections	#AllClusterBuckets	ClusterBucketArray
	#AllDeletedUsers	IdentitySet
	#AllGroups	SymbolKeyValueDictionary
	#AllKerberosPrincipals	SymbolKeyValueDictionary
	#AllUsers	UserProfileSet
	#ConfigurationParameterDict	SymbolKeyValueDictionary
	#ErrorSymbols	SymbolDictionary
	#GemStoneError	SymbolDictionary
	#GemStone_Portable_Streams	SymbolDictionary
	#GemStone_Legacy_Streams	SymbolDictionary
	#Globals	SymbolDictionary
	#InstancesDisallowed	IdentitySet
	#LegacyErrNumMap	Array
	#NotTranloggedGlobals	SymbolDictionary
Other Customer-usable Globals	#DeprecationEnabled	Symbol
	#GciStructsMd5	String
	#GciTsStructsMd5	String
	#GsDefaultIndexOptions	GsIndexOptions
	#IcuLibraryVersion	String
	#StringConfiguration	Class
	#Transcript	TranscriptStreamPortable

Table A.1 Initial Contents of the Globals Dictionary (Continued)

	Key	The object's class
GemStone Internal Objects	#AsciiCollatingTable	ByteArray
	#ConversionReservedOopMap	Array
	#ConversionStatus	Array
	#DoubleByteAsciiCollatingTable	DoubleByteString
	#FdcResults	UndefinedObject
	#GcCandidates	UndefinedObject
	#GcCandidatesCount	UndefinedObject
	#GcHints	UndefinedObject
	#GcWeakReferences	Array
	#GsCompilerClasses	SymbolDictionary
	#GsIndexingObjectSecurityPolicy, #GsIndexingSegment	GsObjectSecurityPolicy
	#ImageVersion	SymbolDictionary
	#ObsoleteClasses	SymbolDictionary
	#QuadByteAsciiCollatingTable	QuadByteString
	#RcBtreeNode	UndefinedObject
	#_remoteNil	UndefinedObject
	#SecurityDataObjectSecurityPolicy, #SecurityDataSegment	GsObjectSecurityPolicy
#SharedDependencyLists	DepListTable	
#VersionParameterDict	SymbolKeyValueDictionary	
plus all kernel classes		

Current TimeZone

Each instance of `DateTime` includes a reference to a `TimeZone` object, which handles the conversion from the internally stored Greenwich Mean Time (GMT, also referred to as UTC or Coordinated Universal Time) and the local time. `TimeZones` encapsulate the daylight savings time (DST) rules, so a given GMT time is adjusted to local time based on `TimeZone` and the specific date. `TimeZones` are also used to calculate the internal stored GMT for newly created `DateTime` instances.

Each session has a current `TimeZone` and a default `TimeZone`, which are used to display times, and in `DateTime` creation when methods that do not explicitly specify the `TimeZone` are used. These are installed as part of application installation or configuration; by default, the GemStone distribution has the `America/Los_Angeles` `TimeZone` installed. This is described in the *GemStone/S 64 Bit Installation Guide*.

GemStone uses the public domain **zoneinfo** database to create `TimeZone`, loading the information from platform and language independent source files. If the rules change for

the TimeZone that your application uses, you must recreate the TimeZone instance from the source files. Depending on the nature of the rules change, you may also need to update references from DateTime instances to the new TimeZone instance, or possibly update the DateTime internal offsets.

There are a number of ways to create TimeZone instances for your application:

- ▶ **From the OS on Solaris or Linux.** On these operating systems, you can create the TimeZone instance based on the current machine configuration using:

```
newTZ := TimeZone fromOS
```

- ▶ **GemStone's time zone database.** Using the interactive script `tzselect`, you can determine the correct time zone descriptor name for your local time zone. With this, you can create the new TimeZone instance using the time zone database provided with GemStone.

```
newTZ := TimeZone fromGemPath: '$GEMSTONE/pub/timezone/etc/zoneinfo/Europe/Zurich'
```

or, if GemStone's time zone database is installed in the default location:

```
newTZ := TimeZone named: 'Europe/Zurich'
```

- ▶ **Your own time zone database.** With the time zone descriptor name for your TimeZone, you can specify the full path to the time zone information.

```
newTZ := TimeZone fromGemPath: yourPath, '/Europe/Zurich'.
```

You must then install this TimeZone instances as the current and default time zone.

Zoneinfo

The widely used public-domain time zone database, **ZoneInfo** or **tz**, contains code and data that records time zone information for locations worldwide. It is updated periodically when boundaries or rules change in any of the represented locations.

Each record in the tz database represents a location where all clocks are kept on the same time as each other throughout the year, coordinating any time adjustments such as DST, and have done so for many years. Locations are identified by continent (or ocean, for islands) and name, which is usually the largest city within the region. For example, `America/Los_Angeles`, `Europe/London`, etc.

tz is provided as text files, which may be compiled into binary files using tz's compilers. GemStone's TimeZone implementation uses the compiled binary form, which is also used by the Solaris and Linux operating systems. GemStone's files are uploaded periodically; to get the latest source files, download from:

```
http://www.iana.org/time-zones
```

The timezone sources may be compiled using the `zic` timezone compiler, which GemStone provides as a convenience (see "zic" on page 385).

Utilities

tzselect, **zdump** and **zic** are public domain, open source utilities that are useful in working with the zoneinfo database. These utilities are provided with the Solaris and Linux operating systems; for the convenience of users on other operating systems, these utilities are provided along with the other zoneinfo database files.

NOTE

These are not GemStone utilities. Support for their use is not provided by GemStone.

You may download the source code for these utilities here:

<http://www.iana.org/time-zones>

Documentation for these utilities is provided as man pages. To read the man pages, add the directory `$GEMSTONE/pub/timezone/man` to the `MANPATH`.

To run these, you may wish to add `$GEMSTONE/pub/timezone/etc` to the executable path.

tzselect

tzselect allows to you interactively select a time zone. The output is suitable as a value for the `TZ` environment variable and GemStone scripts.

You may need to set the environment variable `$TZDIR` to `$GEMSTONE/pub/timezone/etc/zoneinfo` (or the path to your zoneinfo database, for this script to work correctly. You may also need to set the environment variable `$AWK`, to any POSIX compliant `awk` program.

For further details on using **tzselect** see the man page.

zdump

```
zdump [-v] [-c cutoffyear] [zonename...]
```

zdump prints time zone information. It prints the current time for each time zone (`zonename`) listed on the command line. The `-v` option will display the entire contents of the time zone database for the given time zone name.

For further details on using **zdump**, including the command line options, see the man page.

zic

```
zic [-s] [-v] [-l localtime] [-p posixrules] [-d directory]
[-y yeartype] [filename...]
```

zic compiles time zone source files. It reads input text in files named on the command line, and creates the time zone binary files.

For further details on using **zic**, including the command line options and the structure of the source code files, see the man page for **zic**.

Environment Variables

This appendix lists the environment variables used by GemStone/S 64 Bit. The list has two parts: variables intended for public use, and variables that are reserved for internal use.

Public Environment Variables

The following environment variables are intended for use by customers. The variable GEMSTONE is required; the others may be useful in particular situations.

GEMSTONE

The location of the GemStone Object Server software, which must be a full path, beginning with a slash, such as `/user3/GemStone-hppa.hpux`.

GEMSTONE_ADMIN_GC_LOG_DIR

The directory location for Admin Gem logs. By default, Admin Gem log files are created in the same directory as the Stone log, which is `$GEMSTONE/data`. You can set this environment variable to specify that Admin Gem log files are created in a different directory.

GEMSTONE_EXE_CONF

The location of an executable-dependent configuration file; see “Creating an Executable Configuration File” on page 288.

GEMSTONE_GLOBAL_DIR

The location for the global GemStone logs and locks file, overriding the default `/opt/gemstone/`. This directory must already exist.

This directory controls visibility between GemStone processes, and must be the same for all GemStone processes that may want to interact with a given repository, including `stone`, `gems`, `topaz`, `statmonitor`, `netldi`, `gslis`, etc. GemStone processes that do not share a common location for `/gemstone/locks`—either `/opt/gemstone/`, `/usr/gemstone/`, or a directory specified by `$GEMSTONE_GLOBAL_DIR`—operate as if they are in independent name spaces.

GEMSTONE_KEEP_LOG

To keep a process's log from being deleted when the process terminates normally, unset this variable in the appropriate script, such as `$GEMSTONE/sys/gemnetobject`.

GEMSTONE_KEYRING_TABS

Set to a list of colon-separated directories containing SSL keys and certificates, for use by **copydbf -V**.

GEMSTONE_LIB

Specifies the directory for the gem and gem dynamic libraries. This is primarily of use in debugging low-level problems, and is used by the `gemnetdebug` script to specify the slow gem and gem dynamic libraries.

GEMSTONE_LOG

The location of system log files for the Stone repository monitor and its child processes. For further information, see "GemStone Process Logs" on page 122.

GEMSTONE_MAX_FD

Limits the number of file descriptors requested by a GemStone process. Normally, most GemStone processes raise their file descriptor limit from the default (soft) limit to the hard limit set by the operating system. Setting this variable to a positive integer sets a lower limit; a value of 0 disables attempts to change the default limit. This does not apply to the shared page cache monitor.

GEMSTONE_NRS_ALL

Sets a number of network-related defaults, including the type of user authentication that GemStone expects. For further information, see "GEMSTONE_NRS_ALL" on page 370.

GEMSTONE_RECLAIM_GC_LOG_DIR

The directory location for all Reclaim Gem logs. By default, Reclaim Gem log files are created in the same directory as the Stone log, which is `$GEMSTONE/data`. You can set this environment variable to specify that Reclaim Gem log files are created in a different directory.

GEMSTONE_SOCKET_DEBUG

If set to any value, this environment variable will cause tracing messages to be written to stdout for `bind()`, `connect()`, `getaddrinfo()` and related socket calls.

GEMSTONE_SPCMON_STARTUP_TIMELIMIT

Internally, GemStone waits for five minutes for the shared page cache to start up and initialize. This environment variable overrides this timeout, and specifies the time, in seconds, that the Stone will wait for the shared page cache to startup before giving up.

GEMSTONE_SSL_LIB_DIR

A directory location in which to look for the SSL shared library.

GEMSTONE_SYMBOL_GEM_LOG_DIR

The directory location for SymbolGem logs. By default, Symbol Gem log files are created in the same directory as the Stone log, which is `$GEMSTONE/data`. You can set this environment variable to specify that Symbol Gem log files are created in a different directory.

GEMSTONE_SYS_CONF

Location of a system-wide configuration file; see “How GemStone Uses Configuration Files” on page 285.

GS_CFTIME

If defined, it should contain a date and time format string that overrides the GemStone LOCALE-based default. See “Localizing timestamps in log files” on page 131.

GS_CORE_TIME_OUT

If **GS_WRITE_CORE_FILE** is defined, this is the number of seconds to wait before a catastrophically failing GemStone/S process writes a core file and terminates – by default, 60 seconds. To determine the cause of a problem, GemStone/S Technical Support needs a stack trace, which is usually written to the process log file prior to the process shutdown.

If you need to derive a stack trace directly from a failing (but not yet terminated) process by attaching a debugger to it, you can set this variable to increase the time available to attach the debugger.

GS_DEBUG_PAM

If this variable is set to any value, PAM debugging information will be printed to stdout.

GS_DEBUG_SHARED_MEM

If this variable is set to any value, the shared page cache monitor process will print extra debugging to its log file.

GS_DEBUG_SSL_LOG_DIR

In a slow or no-op build only, not available in normal (fast) builds for security reasons. If this variable is set to a directory, a process that logs in RPC will write output of SSL calls made during to a file named `GsSslDebug_<pid>.log` in the specified directory. This file may get very large.

GS_DEBUG_VMGC_MKSW_MEMORY_USED_SOFT_BREAK

At the end of each mark/sweep, if the percent of memory used is greater than the threshold specified by this variable, a SoftBreak (error 6003) is generated, and the threshold is raised by 5 percent. We suggest a setting of 75%.

GS_DEBUG_VMGC_MKSW_PRINT_STACK

The mark/sweep count at which to begin printing the Smalltalk stack at each mark/sweep.

For this and all other **GS_DEBUG_VMGC_*** environment variables, the printout goes to the "output push" file of a linkable Topaz (`topaz -l`) session, for use in testing your application. If that file is not defined, the printouts go to standard output of the session's gem or topaz -l process.

GS_DEBUG_VMGC_MKSW_PRINT_C_STACK

The mark/sweep count at which to begin printing the C stack at each mark/sweep. This variable is very expensive, consuming 2 seconds plus the cost of `fork()` for each printout.

GS_DEBUG_VMGC_PRINT_MKSW

The mark/sweep count at which to begin printing mark/sweeps.

GS_DEBUG_VMGC_PRINT_MKSW_MEMORY

The mark/sweep count at which to begin printing detailed memory usage (20 lines) for each mark/sweep.

GS_DEBUG_VMGC_PRINT_MKSW_MEMORY_USED

Specifies when Smalltalk stack printing starts as the application approaches OutOfMemory conditions. At the end of each mark/sweep, if the percent of memory used is greater than the threshold specified by this variable, the mark/sweep is printed, the Smalltalk stack is printed, and the threshold is raised by 5 percent. In a situation producing an OutOfMemory error, you should get several Smalltalk stacks printed in the Gem log file before the session dies. The default setting is 75%.

GS_DEBUG_VMGC_PRINT_SCAV

The scavenge count at which to begin printing scavenges. Once this takes effect, all mark/sweeps will also be printed. Be aware that printing scavenges can produce large quantities of output.

GS_DEBUG_VMGC_PRINT_TRANS

Print transaction boundaries (begin/commit/abort) in the log file.

GS_DEBUG_VMGC_SCAV_PRINT_STACK

The scavenge count at which to begin printing the Smalltalk stack at each scavenge. Be aware that this print activity can produce large quantities of output.

GS_DEBUG_VMGC_SCAV_PRINT_C_STACK

The scavenge count at which to begin printing the C stack at each scavenge. This variable is very expensive, consuming 2 seconds plus the cost of fork() for each printout.

GS_DEBUG_VMGC_VERBOSE_OUTOFMEM

Automatically call the primitive for `System class>>_vmPrintInstanceCounts:0` when an OutOfMemory error occurs, and also print the Smalltalk stack. (For details about this method, see the comments in the image.) This applies to each Gem or linkable Topaz (topaz -l) process that you subsequently start.

GS_DEBUG_VMGC_VERIFY_MKSW

The mark/sweep count at which to begin verifying object memory before and after each mark/sweep.

GS_DEBUG_VMGC_VERIFY_SCAV

The scavenge count at which to begin verifying object memory before and after each scavenge. Once this takes effect, `GS_DEBUG_VMGC_VERIFY_MKSW` will also be in effect. Be aware that this activity uses significant amounts of CPU time.

GS_DISABLE_CHARACTER_TABLE_LOAD

Any value disables the loading of any customized Extended Character Set Character Data Tables. This feature is deprecated; see the *Programming Guide* for details.

GS_DISABLE_KEEPALIVE

A non-empty string disables the network keepalive facility. For further information about keepalive, see “Disrupted Communications” on page 88

.GS_DISABLE_WARNING

A non-empty string disables a warning that GemStone is using `/opt/gemstone` instead of `/usr/gemstone` for log and lock files when both directories exist. Use of

`/usr/gemstone` is only for compatibility with previous releases; the default location is `/opt/gemstone`.

GS_DISABLE_SHMDT

Disables the system call to `shmdt()` made on cache detach during logout.

GS_DISABLE_SIGNAL_HANDLERS

When this environment variable is enabled in the environment for a gem process, by setting it to any value, this gem sessions will not attempt to handle a SIGSEGV, SIGBUS or SIGILL signal, but will shut down immediately. It will not generate a core nor print C stacks. This avoids side effects with user action or client Smalltalk code.

GS_GSLIST_TIME_FORMAT

This can be set to UNIX-style date format string, to allow the output of the `gslis` utility to be displayed in a parse-able format.

GS_FORCE_CLEAN_LOG_FILE_DELETE

When set, delete GemStone log files on clean exit; affects all processes except for Stone or NetLDI logs, which are never deleted. Logs for processes that exit with errors are not affected.

GS_KEEP_ALL_LOGS

Ensure that no GemStone process log files are deleted, overriding any individual process settings for `GEMSTONE_KEEP_LOG`.

GS_PAGE_MGR_PRINT_REMOTE_STACKS

If this variable is set, if a remote cache page server becomes stuck, the page manager will request that the remote cache page server print its call stack to its log file.

GS_SOCKET_SEND_BUF_SIZE

Sets the size of the socket send buffer for the socket between the Gem and GCI client for an RPC Gem.

GS_SOCKET_RECV_BUF_SIZE

Sets the size of the socket receive buffer for the socket between the Gem and GCI client for an RPC Gem.

GS_WRITE_CORE_FILE

By default, core files are not created when a fatal error occurs. (The C level stack trace is written to the process log file prior to the process shutdown.) You can set this environment variable if you need a core file.

upgradeLogDir

The location for log files produced during the upgrade of a repository for a new version of GemStone.

System Variables Used by GemStone

GemStone uses the following system variables that exist for other purposes:

EDITOR	Used by Topaz to determine which editor to invoke.
PATH	The search path of locating executable files.
SHELL	Used to determine what shell to use for an exec, such as by <code>System class>>performOnServer:.</code>

Reserved Environment Variables

The following environment variables are reserved for internal use. Customers should not define these variables for use with GemStone unless specifically instructed to do so. Please refrain from using these variables for other purposes.

GCIRTL_BASELIBNAME

GEMSTONE*

All environment variable names beginning with "GEMSTONE" other than those above are reserved.

GS_*

All environment variable names beginning with "GS_" other than those above are reserved.

NT_PARENT_PID

gs64ldi

runpgsvr

Object State Change Tracking

A.1 Overview

GemStone transaction logs (tranlogs), which provide a way to recover all changes made to the repository in the case of repository crashes, or allow warm standbys to apply changes made to a primary repository, include detailed records of all changes in an encoded and compressed form. Converting the tranlog information to human-readable form, and analyzing this output, provides invaluable information for debugging and testing. It allows us to determine exactly what changes have been made to any of the objects in the repository over time. The tools used to perform this have been used internally to GemStone for many years, and have been provided to customers on occasion when needed to analyze specific application problems.

The ability to track changes to objects may be useful for customers who need to identify details on changes that have been made to application data objects. For this reason, we are making these scripts available as part of the GemStone product. Additional information has been added to the tranlogs to allow tracking of the specific user or machine that originated the object changes.

The scripts used to perform tranlog analysis are located in the `$GEMSTONE/bin/` directory and are named:

`printlogs` — to output the complete contents (optionally filtered) of selected tranlogs in human-readable form.

`searchlogs` — to search all tranlogs in a directory for selected OOPs (Object Oriented Pointers, or Object Ids), and output the matching entries (optionally filtered) in human readable form.

A GemStone repository performs many automatic operations, including things like garbage collection and checkpoints, that are transparent to end users. The tranlogs, of course, must contain records of any changes made by these operations. Complete details on everything that tranlogs may contain is beyond the scope of this documentation. The information provided here is intended to allow the use of the tranlog analysis scripts to locate and identify the details of changes to application objects.

Object oriented design's principle of encapsulation allows you to hide internal object complexity. However, to understand the data recorded in the tranlogs, you must have a detailed understanding of the actual structure of the objects. This includes both your own application classes, and the classes that are part of GemStone Smalltalk.

Also, note that since the tranlog analysis scripts are general purpose, used for a wide variety of purposes in which a detailed record of internal repository operation is required, the scripts may output much more information than is necessary for tracking object state changes. You may need to ignore this extraneous information as you perform your analysis.

A.2 Tranlog Analysis Scripts

Script Prerequisites

The environment variable `$GEMSTONE` must be set, and the `$GEMSTONE/bin/` directory must be in your executable path.

The scripts perform the analysis on tranlogs that are in the current working directory. If you are using raw partitions for your tranlogs, locate disks on the file system with adequate space for both the tranlogs themselves, and the script output files, which may be larger than the original tranlogs. Use `copydbf` to copy the tranlogs from the raw partition to the file system. For more information on the `copydbf` utility, see Appendix B, "GemStone Utility Commands".

Output

Output from the scripts goes directly to `stdout`. To preserve the output and allow it to be used in later steps of analysis, redirect this; for example:

```
$> printlogs tranlog1.dbf > tranlog1.out
```

Note that these scripts can produce very large amounts of output, so make sure that you have adequate disk space.

In some cases the resulting files may be too large for unix text editors such as `vi` or `emacs` to open. You may find it necessary to use the unix `split` utility to break up very large output files into more manageable chunks.

Tranlog Assumptions

By default, the tranlogs are assumed to be named using the GemStone convention `tranlogNNN.dbf`. If you are using a different naming convention, you can override this by setting the environment variable `$GS_TRANLOG_PREFIX` to the prefix you are using.

The scripts use the creation date of the tranlog file to determine the order in which the tranlogs are analyzed. If you copy or manipulate the tranlogs in a way that changes the creation date, this may cause the analysis to be done out of order. The output will warn of this with the message:

```
*** Warning:  scanning tranlogs out of order
```

Filter Criteria

The scripts both allow you to filter the results, to locate entries that are related to a particular `UserId`, `GemHost`, or `ClientIP`.

UserId – The `userId` (user name) of the `UserProfile` associated with this session: `DataCurator`, `SystemUser`, etc. The filter keyword is `user`.

GemHost – The name or IP address of the host machine running the gem process. For a linked session, which links the gem into the client, this is the same machine as the client.

If the gem is running on the same machine as the stone, use the name of the host machine. Otherwise, if the gem is on a different machine than the stone, use the IP address of the remote machine.

The filter keyword is `host`.

ClientIP – The IP address of the host machine running the client process.

For an RPC session, this is the machine running the client application. Clients may be `topaz -r`, `GemBuilder` for Smalltalk, or `GemBuilder` for C applications. For a linked session, this is the machine running the linked client/gem (the same machine as the `GemHost`). However, the `ClientIP` is always the IP address, even if it is on the same machine as the stone.

The filter keyword is `client`.

Effective UNIX user ID – The integer that is the effective UNIX user id of the gem process.

The filter keyword is `eid`.

UNIX user ID – The integer that is the real UNIX user ID of the gem process.

The filter keyword is `ruid`.

effective UNIX user name – The effective UNIX user name of the gem process.

The filter keyword is `eidstr`.

UNIX user name – The real UNIX user name running the gem process.

The filter keyword is `ruidstr`.

process ID – The integer process id (PID) of the gem process.

The filter keyword is `gempid`.

session ID – The integer session id of the session within GemStone.

The filter keyword is `sessionid`.

WARNING

*Information about `UserId`, `GemHost`, and `ClientIP` are derived from a **Login** tranlog entry created when a session first logs in. This entry associates the `UserId/GemHost/ClientIP` with a particular `sessionID`, which is then used as a key for subsequent tranlog entries. If you start analysis from a later tranlog which does not include this **Login** entry, these fields will be left blank for that session,*

*and printouts/searches using filters based on these fields will not locate any results. Likewise, scanning through tranlogs out of order may result in the wrong **Login** entry being associated with a given sessionID. This would set UserId/GemHost/ClientIP incorrectly for that particular session, and produce incorrect results when filtering.*

printlogs

This script prints out the contents of one or more tranlogs in the current working directory in a human-readable form.

Warning

This script produce a very large amount of output, which (unfiltered) will exceed the size of original tranlog/s, and depending on the contents may be twice as large as the original tranlogs. Consider disk space, the use of filters, and restricting the set of tranlogs before running this script. Use caution in including the `full` keyword.

Usage:

```
printlogs [<filters>] [full] [all] [<tlogA> ... <tlogZ>]
```

If `<filters>` are specified, only print out the tranlog entries that match the specified criteria. Filters may be combined. Possible filters are:

```
user <userId> - Filter by the userId (the user name) of the GemStone UserProfile
host <hostnameOrIP> - Filter by gem/topaz process host or IP address
client <N.N.N.N> - Filter by client IP Address
euid <integer> - Filter by gem's effective UNIX user ID
ruid <integer> - Filter by gem's real UNIX user ID
euidstr <string> - Filter by gem's effective UNIX user name
ruidstr <string> - Filter by gem's real UNIX user name
gempid <integer> - Filter by gem's process ID
sessionid <integer> - Filter by gem's session ID
```

`full` – Produce more detailed information. WARNING: this produces much larger output results.

`all` – Print out the contents of all tranlogs in the current working directory.

Examples

To print out the entire contents of all tranlogs in this working directory:

```
printlogs all
```

To print out all entries in a selected number of tranlogs (note that tranlogs in the sequence must be contiguous):

```
printlogs tranlog5.dbf tranlog6.dbf tranlog7.dbf
```

To print out all tranlog entries for the user DataCurator in any tranlog:

```
printlogs user DataCurator all
```

To print out detailed information for all entries in tranlog5.dbf for the user DataCurator:

```
printlogs full user DataCurator tranlog5.dbf
```

searchlogs

This script prints out tranlog entries associated with particular OOP values, according to the arguments in the command line. All tranlogs in the current working directory are scanned.

Usage:

```
searchlogs [<filters>] <oopA> ... <oopB>]
```

If <filters> are specified, only print out the tranlog entries that match the specified criteria. Filters may be combined. Possible filters are:

```
user <userId> – Filter by the userId (the user name) of the GemStone UserProfile
host <hostnameOrIP> – Filter by gem/topaz process host or IP address
client <N.N.N.N> – Filter by client IP Address
euid <integer> – Filter by gem's effective UNIX user ID
ruid <integer> – Filter by gem's real UNIX user ID
euidstr <string> – Filter by gem's effective UNIX user name
ruidstr <string> – Filter by gem's real UNIX user name
gempid <integer> – Filter by gem's process ID
sessionid <integer> – Filter by gem's session ID
```

When using more than one filter, you must list the filters in the listed order.

Examples

To print out all entries involving OOP 1234 and OOP 5678:

```
searchlogs 1234 5678
```

To print out all entries involving OOP 1234 performed by DataCurator:

```
searchlogs user DataCurator 1234
```

To print out all entries involving OOP 1234 and OOP 5678 performed by DataCurator while logged in from client machine 10.20.30.40:

```
searchlogs user DataCurator client 10.20.30.40 1234 5678
```

A.3 Tranlog Structure

In order to make sense of the output from the scripts, you need a basic understanding of how tranlogs are structured.

GemStone transaction logs consist of a sequence of **tranlog records**. Tranlog records are written on **physical pages** of 512 bytes (note that this is different from the larger page size used for extents); a tranlog record may extend over more than one page, but its size is always a multiple of 512 bytes.

Each tranlog record contains one or more **tranlog entries** (sometimes referred to internally as logical records). The tranlog entries contain the information of interest - the actual changes to objects in the repository (and any other repository operations).

Output from the scripts will include header information for the tranlog record (see Example A.1), followed by data from each of the tranlog entries held by that tranlog record.

Example A.1 Tranlog Record Header Output

```
Dump of record 11 hdr.pageId 42949672962 , Kind=(16)Tran Log Record
thisRecordId:(file:2 rec:11) previousRecordId:(file:2 rec:10)
recordSize: 1024 numLogicalRecs: 2 fileCreationTime: 1297200655
```

Tranlog records are identified by the pageId that they begin on. Since a tranlog record may extend over multiple pages, there may be a gap in the sequence of record ids in the output. For example, the first tranlog record in Example A.2, record 7, has a recordSize of 1536 (three 512-byte physical pages), and so the next tranlog record will be 10.

Example A.2 Gap in Tranlog Record Sequence

```
Dump of record 7 hdr.pageId 25769803778 , Kind=(16)Tran Log Record
thisRecordId:(file:2 rec:7) previousRecordId:(file:2 rec:6)
recordSize: 1536 numLogicalRecs: 2 fileCreationTime: 1297200655
```

```
2.7.0 BeginData session: 4 beginId:(100979 1)
  numObjs:3 pad1:0 pad2:0
  2195969 240385 20423937
```

```
2.7.1 Commit session: 4 beginId:(100979 1) crPage: 1039 commitSeq: 0.158
timeWritten: 02/08/11 13:31:52 PST
  seqType:normal beginLogRecord:(file:2 rec:7)
```

```
-----
Dump of record 10 hdr.pageId 30064771074 , Kind=(16)Tran Log Record
thisRecordId:(file:2 rec:10) previousRecordId:(file:2 rec:7)
recordSize: 512 numLogicalRecs: 3 fileCreationTime: 1297200655
```

Tranlog Entries

Each tranlog entry contains a unique identifying code, a descriptive entry type, and information on the session that originated the tranlog entry.

The identifying code consists of three numbers in the form:

AAA.BBB.CCC

where:

AAA – the fileId of the tranlog holding the entry

BBB – the pageId for the tranlog record holding the entry

CCC – the entryId: the number of the entry within the tranlog record

Each tranlog entry is of a particular type, according to the event that it represents and the information it contains. Types are indicated by short descriptive terms such as **Login**, **Abort**, **Commit**, and **Data**. There are a large number of entry types, most of which are not important for tracking object state changes and can be ignored (for example, a **Checkpoint** entry. The ones that are important are documented below.

Each tranlog entry is associated with an Integer sessionID. SessionsIDs are unique to a specific session at any point in time. However, when a session logs out, the sessionID becomes available again for reuse by a new session logging in, so over a period of time, a sessionID may be used by a number of different sessions.

A sessionID of zero is used for tranlog entries generated by the stone.

If the entry was not originated from the stone (that is, the tranlog entries sessionID is not 0), the tranlog entry header includes more information about the session: the UserId, the GemHost, and the ClientIP address. These are described in more detail under “Filter Criteria” on page 395.

Example A.3 Example Tranlog Entry

```
2.3.4 Login session: 4 beginId:(100973 1) userProfileOop: 208385
timeWritten: 02/08/15 13:30:56 PST userId: SymbolUser gemhost: myhost
clientIP: 10.20.30.40 processId: 12663 rUId: 531 eUId: 531 realU: gsadmin
effU: gsadmin
```

```
2.3.5 StartSymbolGem session: 4 beginId:(100973 5)
```

Example A.3 shows that is in the tranlog with fileId 2 (by the default naming convention, `tranlog2.dbf`), it is in the third physical page and in tranlog record 3, and these are the fourth and fifth tranlog entries in tranlog record 3.

These entries are of the tranlog entry types `Login` and `StartReclaimGcGem`. The session is logged in as the user named `SymbolUser`; the gem session is executing on the same machine as the stone, a machine named `myhost`; the client is executing on a machine with the IP address `10.20.30.40`; and the OS process is owned by the OS userid `gsadmin`. (`beginId` contains transaction tracking information that you can ignore).

Other information will follow this basic data, depending on the type of entry.

Tranlog Entry Types

There are a large number of tranlog entry types. Most of these are not relevant to tracking object state history - they record internal operations of the system, such as garbage collection or checkpoints. These tranlog entry types are not documented, although their functions can often be deduced by their names.

Below are the entries important for tracking object state history:

Login

A new session is logging into GemStone. As mentioned earlier, this entry logs the `UserId`, `GemHost`, `ClientIP` and other data for this particular sessionID. If you start analysis on a tranlog that follows this event, these fields will be left blank for that session.

For example, if session 7 logs in while `tranlog4` is in effect, and logs out when `tranlog5` is in effect, and you begin analysis on `tranlog5`, entries for this session will not contain any session detail information. If sessionID 7 is reused by a new session logging in later during `tranlog5`, that login will be recorded in `tranlog5`, and subsequent entries for this new session will be displayed properly.

AbortLogout

This entry is written when a session logs out or the Stone detects that the session has been killed. This entry is not flushed to the transaction log until a commit occurs.

BeginData

Data

BeginStoreData

StoreData

GemStone uses the above four entry types for recording new or changed object data. The basic entry information is followed by additional information about the changes, including the OOP values of the changed objects. Using the optional “full” flag in the `printlogs` script will cause the output to include additional information on the exact changes made.

Commit

All the changes recorded in earlier **BeginData**, **Data**, **BeginStoreData**, or **StoreData** entries are now officially committed to the repository.

Abort

Any changes recorded earlier in this transaction are discarded.

BreakSerialization

This entry indicates that a transaction conflict occurred during an attempt to commit. Any changes recorded earlier in this transaction are not yet permanent. This event is usually followed by an **Abort** entry, although it's possible that the next event seen for this sessionID is a **Login**, if the original session logged out and a new session reuses the sessionID.

Very Large Objects

GemStone is designed so that all objects will fit on a single page of 16384 bytes. This means that a byte object can be no larger than about 16000 bytes (since page header information uses some space), and pointer objects can only have about 4000 elements. GemStone internally represents objects larger than this as a tree structure, where the root node object references 2 or more leaf node objects, which then reference the actual elements of the collection object. Extremely large objects, such as large collections, may have internal branch nodes, if the number of leaf objects needed exceeds 4000.

This internal structure is usually transparent to the user. So, for example, you may create and manipulate an Array containing 20,000 elements as if it was a single large object, while the actual representation is a root object that references five leaf objects, each containing a 4000-element chunk of the array. While this makes application development with GemStone much simpler, the entries in the tranlogs reflect the actual implementation; you will need to be aware of this to understand tranlog output relating to collections larger than ~4000 object references or ~16000 bytes. Adding an element to the large Array, for example, may mean the tranlog entry includes a change to an instance of LargeObjectNode (the leaf node object), rather than a change to the Array itself.

Full vs. Normal Mode

When using the `printlogs` script, you can optionally specify “full” mode to get more details on the changes made to objects in the repository. But this will greatly increase the size of

the resulting log files. For example, using normal mode on our test tranlogs generated a log file that contained an entry that looked like this:

Example A.4 Tranlog entry, normal mode

```
2.11.0 BeginData session: 5 beginId:(100978 1)
      numObjs:3 pad1:0 pad2:0
      25567233 8532993 25880577
```

which tells you that three objects were created or modified during this event, with oops 25567233, 8532993, and 25880577.

Using “full” mode will produce a much more detailed listing for this event:

Example A.5 Tranlog entry, full mode

```
2.11.0 BeginData session: 5 beginId:(100978 1)
      numObjs:3 pad1:0 pad2:0

objId 25567233 class 114177 segId:9 len 14 gcSz 14 psize 136 bits 0x1 page
1152
Oop values: 50(sI 6) 8532993 5042177 25881601 17979137 26620161
20272897 25801985
8: 20322817 25559553 20324097 25559297 20423937 25869569

objId 8532993 class 111361 segId:9 len 14 gcSz 14 psize 136 bits 0x1 page
1152
Oop values: 162(sI 20) 122(sI 15) 20002(sI 2500) 42(sI 5) 20 18224129 20
25567233
8: 20 25566977 20 25801729 20 25566721

objId 25880577 isNewObj=1 class 73985 segId:9 len 23 gcSz 23 psize 208
bits 0x241 page 1152
Oop values: 18(sI 2) 18(sI 2) 20 2(sI 0) 25870081 25871105 20 20
8: 20 20 20 20 20 20 20 20
16: 20 20 20 20 20 20 20
```

Note that there is now a description of each individual created or modified object, containing these fields:

`objId`: The OOP of this object
`isNewObj`: A Boolean indicating if the object is a newly created object.
`class`: The OOP of the class of this object
`segId`: The id of the GsObjectSecurityPolicy (formerly Segment) associated with this object
`len`: The logical size of this object (in bytes for a byte object, OOPs for an OOP object)

gcSz: The logical size of this object (in bytes for a byte object, OOPs for an OOP object)

psize: The physical size of this object on disk in bytes (including 20 bytes of object header information)

bits: The format bits for this object (internal GS use)

page: The extent page that this object is written on

Bytes: The actual bytes that make up this object (if a byte object)
[or]

Oop values: The actual OOPs that make up this object (if an OOP object)

If the Oop values contain more than eight elements, they are broken into lines of eight items, each of which is prefixed by a counter. For example, an Array of 30 items might look like this:

```
objId 39056641 isNewObj=1 class 66817 segId:9 len 30 gcSz 30
psize 264 bits 0x201 page 1005
Oop values: 39056385 39056129 39055873 39055617 39055361
39055105 39054849 39054593
8: 39054337 39054081 39053825 39053569 39053313 39053057
39052801 39052545
16: 39052289 39052033 39051777 39051521 39051265 39051009
39050753 39050497
24: 39050241 39049985 39049729 39049473 39049217 39048961
```

Bytes are broken up similarly, but the sections are 60 bytes rather than 8. For example, the source code string for the name: method might look like this:

```
objId 38711809 isNewObj=1 class 74753 segId:9 len 91 gcSz 0 psize
120 bits 0x280 page 1097
Bytes: name: newValue

"Modify the value of the instance variabl
61: e 'name'."
name := newValue
```

A.4 Example of Tranlog Analysis

For this example, we created a simple database containing some Employee information and performed some simple operations creating and modifying these Employee objects.

The structure of classes associated with the Employee data is as follows:

Employee:

```
name - a Name object
age - a SmallInteger
address - an Address object
```

Name:

```
last - a String object
first - a String object
middle - a String object
```

Address:

```
addr1 - a String object
addr2 - a String object
city - a String object
state - a String object
zip - a SmallInteger
```

After having User1 create five Employee objects (with associated Name and Address objects), we then had User1 and User2 make some minor changes to one of the Employees:

- ▶ User2 incremented the age after a birthday.
- ▶ User1 changed the address after a move.

When completed, we had two tranlogs, tranlog2.dbf and tranlog3.dbf.

Tracking Changes to an Employee

Let's say we want to examine the change history of a particular Employee. Using the method #asOop, we find the OOP of the Employee object of interest is 38808321.

```
topaz 1> printit
| myEmployee |
myEmployee := <code to locate employee object> .
myEmployee asOop.
%
38808321
```

The data composing an Employee is contained in subobjects (such as address), as well as directly (such as age). So, we will also need to track changes to these subobjects. Again using #asOop, we find that the OOP of the Name object associated with this Employee is 155073, and that the OOP for the Address object is 155069.

```
myEmployee name asOop
%
38808577
myEmployee address asOop
%
38808833
```

You can use #asOop on any persistent object in the repository. For example,

```
73 asOop
%
586
nil asOop
%
20
```

We can now search our tranlogs for any events involving these objects. Using the command:

```
$> searchlogs 38808321 38808577 38808833
```

results in the following output:

Example A.6 seachlogs output for Employee

Searching for oops: 38808321 38808577 38808833
Searching tranlogs:

```
tranlog2.dbf
tranlog3.dbf
```

... header material omitted ...

```
2.118.0 BeginData session: 5 beginId:(101033 0) newobj 38808321 cls
39017217 onPage 1081, newobj 38808577 cls 38726401 onPage 1081, newobj
38808833 cls 38715649 onPage 1081
```

```
2.123.0 Commit session: 5 beginId:(101033 0) timeWritten: 02/08/11
14:43:42 PST
```

```
2.215.0 BeginData session: 6 beginId:(101664 1) object 38808321 cls
39017217 onPage 871,
```

```
2.215.1 Commit session: 6 beginId:(101664 1) timeWritten: 02/08/11
16:14:48 PST
```

```
3.22.0 BeginData session: 8 beginId:(101779 1) object 38808833 cls
38715905 onPage 900,
```

```
3.22.1 Commit session: 8 beginId:(101779 1) timeWritten: 02/08/11
16:32:58 PST
```

From this output, we can see that in tranlog entry 2.118.0, Session 5 made changes to all three objects (in this case, when the Employee and associated subobjects were first created). In entry 2.215.0, session 6 made a change to the Employee, and then later in entry 3.22.0, session 8 made a change to 38808833, the Address object.

Note that the **BeginData** entries are each followed by a **Commit**. You should always confirm that a **BeginData/Data/BeginStoreData/StoreData** of interest is followed by a **Commit**. If it doesn't then the reported event was not made persistent in the repository.

Changed vs. new objects

In the above example, while the field of an Address object changed, the Address object itself was the same (had the same OOP). Depending on how the Smalltalk application is written, this may not always be the case. If application that was initiating these changes created a new Address object, and assigned the Employee's address instance variable to this new object, then the Employee object would reference a new OOP, rather than OOP 38808833. This would make the analysis somewhat different. For example, in the initial stage of the analysis when you look up the OOP of the Address object in your application, you would find the new OOP rather than the original OOP. Looking back in time, you would see when this Address object was created and assigned to the Employee instance.

Details of Changes to an Employee

Having used the `searchlogs` script to get a general idea of which tranlogs are of interest, you can now use the `printlogs` script to get more details.

For example, you might want more details on the creation of Employee object 38808321 and its associated subobjects 38808577 and 38808833 in entry 2.118.0. The "2" in "2.118.0" indicates that `tranlog2.dbf` is the tranlog of interest. The command:

```
$> printlogs tranlog2.dbf
```

will generate a condensed listing of all events in `tranlog2.dbf`. By searching the resulting file for the entry number 2.118.0 you can find the relevant entry:

Example A.7 Employee modification

```
3.61.0 BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(134 0)
  clusterId: 1, extentId: 0 numObjs:35

145021 155041 155045 155049 155053 155069 155073 155077 155089
155093 155097 155101 155121 155273 155285 155317 156589
156597 156609 156613 156661 156689 156729 156733 156749
156825 156829 156833 156837 156857 156861 156885 179629
180045 180653
```

This shows the oops of *all* objects created during this event. If you want to see more details on the actual changes made, use the "full" argument in the `printlogs` command:

```
$> printlogs full tranlog3.dbf
```

This will produce a more detailed listing of all events. For tranlog entry 3.61.0, you'll find:

Example A.8 Example Employee modifications, output in full mode

```
2.118.0 BeginData session: 5 beginId:(101033 0)
  numObjs:11 pad1:0 pad2:0

[details for other objects omitted]

objId 38808321 isNewObj=1 class 39017217 segId:9 len 3 gcSz 3 psize 48
bits 0x201 page 1081
Oop values: 38808577 306(sI 38) 38808833

objId 38808577 isNewObj=1 class 38726401 segId:9 len 3 gcSz 3 psize 48
bits 0x201 page 1081
Oop values: 38809089 38809601 20

objId 38808833 isNewObj=1 class 38715649 segId:9 len 5 gcSz 5 psize 64
bits 0x201 page 1081
Oop values: 38809857 20 38810113 38810369 777714(sI 97214)

objId 38809089 isNewObj=1 class 74753 segId:9 len 7 gcSz 0 psize 32 bits
0x280 page 1081
```

Bytes: Patrick

```
objId 38809601 isNewObj=1 class 74753 segId:9 len 5 gcSz 0 psize 32 bits
0x280 page 1081
```

Bytes: Ohara

```
objId 38809857 isNewObj=1 class 74753 segId:9 len 13 gcSz 0 psize 40 bits
0x280 page 1081
```

Bytes: 2556 Fir Blvd

```
objId 38810113 isNewObj=1 class 74753 segId:9 len 7 gcSz 0 psize 32 bits
0x280 page 1081
```

Bytes: Ashford

```
objId 38810369 isNewObj=1 class 74753 segId:9 len 2 gcSz 0 psize 32 bits
0x280 page 1081
```

Bytes: OR

So, for the Employee object 38808321, we find:

```
objId 38808321 isNewObj=1 class 39017217 segId:9 len 3 gcSz 3
psize 48 bits 0x201 page 1081
```

```
Oop values: 38808577 306(sI 38) 38808833
```

Indicating that this is an instance of class 39017217, the Employee class, which has three instance variables: name, age, and address. By position, we can identify the data in the instance variables:

name: 38808577 - the OOP of an instance of Name, found later in the entry

age: 306 - the OOP of the SmallInteger (sI) 38

address: 38808833 - the OOP of a instance of Address, found later in the entry

Looking at the Name object 38808577 we find:

```
objId 38808577 isNewObj=1 class 38726401 segId:9 len 3 gcSz 3
psize 48 bits 0x201 page 1081
```

```
Oop values: 38809089 38809601 20
```

Indicating that this is an instance of the class with OOP 38726401 (Name). Name contains three instance variables, last, first, and middle. By position, we see the data is:

last: 38809089 - the OOP of a String, described below

first: 38809601 - the OOP of a String, described below

middle: 20 (the OOP of nil) - in this example, no middle name was set

For the last name object 38809089, we find:

```
objId 38809601 isNewObj=1 class 74753 segId:9 len 5 gcSz 0 psize
32 bits 0x280 page 1081
```

Bytes: Ohara

Indicating the last name is the string "Ohara".

By a similar process you can follow the trail of objects to examine the structure of other subobjects in the Name and Address objects.

A.5 Further Analysis

Class Operations

To find all objects created or modified that belong to a particular class, first generate `printlogs` output in full mode of the tranlogs of interest. Each time an object of that class is created or modified, the full tranlog entry includes the line

```
class <OOP>
```

Use the unix `grep` command to find all references to the OOP of the class.

For example, to find all creation or modification of any instance of our example class `Employee` in the `printlogs` full output.

Since the `Employee` class is OOP 39017217, execute the `grep` command:

```
$> grep "class: 39017217" tranlogfull.txt
```

this will produce output of the form:

```
objId 38804481 isNewObj=1 class 39017217 segId:9 len 3 gcSz 3
psize 48 bits 0x201 page 1087
objId 38808321 isNewObj=1 class 39017217 segId:9 len 3 gcSz 3
psize 48 bits 0x201 page 1081
objId 40062465 isNewObj=1 class 39017217 segId:9 len 3 gcSz 3
psize 48 bits 0x201 page 1317
objId 38808321 class 39017217 segId:9 len 3 gcSz 3 psize 48 bits
0x1 page 871
objId 38808321 class 39017217 segId:9 len 3 gcSz 3 psize 48 bits
0x1 page 900
```

This gives the first line from the entry creating/modifying the object belonging to that class. From this, you can use other commands to search for and/or track the history of these objects.

Deleted Objects

An object-oriented system doesn't actually delete objects; objects cease to be referenced and are eventually garbage-collected. Noting the removal of an object requires examining the references to that object (such as from a collection) and identifying when the referencing object was modified in such a way that the object of interest is no longer referenced.

Meanwhile, as you examine the `printlogs` output, you may find references to the OOP of a dereferenced object in garbage collection tranlog entries.

Managing Volume

As noted above, the `printlogs` produce a very large amount of output. GemStone tranlogs may be multiple GB in size. The output of `printlogs` in normal mode will be somewhat larger than the original tranlog (the `printlogs` output, being human readable, is less dense). The output from this script in full mode is much larger.

To manage the volume:

- ▶ Avoid configuring your system with very large tranlogs.
- ▶ Ensure that you have plenty of disk space available before beginning analysis.

- ▶ Print only the tranlogs containing data you need. Use the `searchlogs` script to identify exactly where the required information is located.
- ▶ Make sure that only the relevant tranlogs are in the current directory; move the unneeded ones elsewhere. However, you must retain a continuous set of tranlogs without gaps in sequence, and you must include the tranlog with the original log entry, in order to have the `UserId` and other information provided.
- ▶ Once you have printed the output, use the UNIX utility `grep -n` to locate the lines of interest, and the UNIX utility `split -l` to break the resulting file up into more manageable size chunks.

Index

A

ad hoc processes 76, 353
adding

- a new user 155
- extent 187
- extent, while running 188
- symbolList to user 162
- transaction log at runtime 205
- user privileges 161

Admin Gem 261

- configuring 276

AIO page server

- defined 25

AllClusterBuckets 379

- see also the *Programming Guide*

AllDeletedUsers 152, 379
AllGroups 153, 379
AllKerberosPrincipals 379
allocation

- of extents, weighted 47
- of objects to new extents 190

AllUsers 146, 379
application, linked vs. RPC 64
assigning privileges to a user 155
audit

- object audit 134
- page audit 132

auth modifier, NRS 78
authentication scheme

- setting to LDAP 175
- setting to UNIX 174

authentication scheme, determining 148

authorization

- and object security policies 149, 159
- list of an object security policy, removing a group from 160

automatic garbage collection 249

B

backups

- checkpoint at start of 215
- compressed 218
- errors in full backup 217
- for warm or hot standby 239
- full backup 211
- GcLock 217
- interactions with garbage collection 217, 218
- making full backup 215–218
- making offline extent snapshot 212
- making online extent snapshot 213–215
- monitoring full backup 218
- of transaction logs 204
- offline extent snapshot 211
- online extent snapshot 211
- performance of full backup 217
- restoring
 - to point in time 235
- restoring from full backup 222–225
- restoring transaction logs into 230–232
- secure 225
- transaction mode and 217
- using copydbf 212
- using operating system facilities 212
- verifying readability 218

with repository online 215
 baseDn: (login authentication) 175
 beginTransaction (System) 195

C

cache statistics
 programmatically access 138
 see also under *statistics*
 setting gem name in 141
 statistics names 139
 user defined session 142
 user-defined global session 142
 cache warming 296, 297, 312, 313, 347
 cacheName
 setting name in cache for gem 141
 ChangeUserId (privilege) 151
 checkpoint
 defined 186
 determining status 213
 frequency of 58
 identifying in transaction logs 339
 operating system backup and 212
 resuming 214
 starting 206
 Stone shutdown and 59
 suspending 213
 clock, system 33
 clustering
 new extents and 190, 191
 restoring backups and 220
 see also the *Programming Guide*
 code
 region of temporary object memory 249
 CodeModification (privilege) 150, 151
 commit
 disable user 166
 re-enable user 166
 test if user is enabled to 166
 commit record
 defined 257
 commit record backlog 194, 195, 327
 defined 257
 committed objects
 in local object memory 249
 communications, disrupted 88
 CompilePrimitives (privilege) 151
 compiler and symbol resolution 151
 compressed backups
 creating 218
 concurrent I/O 316
 configuration

 access at run time
 Gem 68
 Stone 55
 extent locations 45
 raw partitions 52
 single-host 64
 transaction logs 49
 tuning 58
 configuration files
 examining parameters from Smalltalk 55, 68
 executable 285
 for server (Stone) 40
 for sessions 65
 naming options 291
 option value errors in 292
 options, warning messages and 291
 printing summary of all options 296
 searching for 286
 syntax errors in 292
 syntax of 291
 system-wide 285
 used by GemStone 286
 configuration options 294–334
 and ConfigurationParameterDict 379
 programmatically access 55, 68
 configurationAt: (System) 55, 68, 332
 configurationAt:put: (System) 332
 ConfigurationParameterDict 379
 continuous restore mode, for hot standby 243,
 245
copydbf command 336
 archiving transaction logs 204
 using with raw partitions 53
 createExtent: (Repository) 188
 createExtent:withMaxSize: (Repository)
 189
 creating
 executable configuration files 288
 extents 49
 transaction logs 49, 201
 current object security policy, exercise caution
 when changing 160
 currentLogDirectoryId (Repository) 205
 currentLogFile (Repository) 205
 currentSessionNames (System) 114, 116, 281
 currentTranlogSizeMB (Repository) 204, 205

D

DataCurator
 and AllUsers system object 379
 and DataCuratorObjectSecurityPolicy object
 378

- described 147
 - DataCuratorGroup 153
 - DataCuratorObjectSecurityPolicy 153, 378
 - Date and time formatting 131
 - DBF_ALLOCATION_MODE 47–49, 190, 191, 219, 294
 - DBF_EXTENT_NAMES 45, 53, 107, 185–191, 294
 - DBF_EXTENT_SIZES 44–47, 187, 197, 295
 - DbfHistory 380
 - DbfOrigin 380
 - DBF_PRE_GROW 47, 189, 295
 - DBF_SCRATCH_DIR 296
 - dead object
 - contrasted with shadow object 257–260
 - defined 257
 - DeadNotReclaimedObjs 275
 - #deadObjsReclaimedCommitThreshold (GcUser parameter) 273
 - DeadObjsReclaimedCount, in monitoring reclaim 275
 - debugging
 - out-of-memory errors 252
 - DecimalMinusInfinity (kernel object) 377
 - DecimalMinusQuietNaN (kernel object) 377
 - DecimalMinusSignalingNaN (kernel object) 377
 - DecimalPlusInfinity (kernel object) 377
 - DecimalPlusQuietNaN (kernel object) 377
 - DecimalPlusSignalingNaN (kernel object) 377
 - default object security policy
 - changing a user's 160
 - defined 146, 149
 - described 146
 - exercise caution when changing 160
 - privilege required in UserProfile 160
 - DefaultObjectSecurityPolicy (privilege) 150
 - #deferReclaimCacheDirtyThreshold (GcUser parameter) 273
 - DeletedUserProfile 152
 - deleting user privileges 161
 - DeprecationEnabled 380
 - descriptionOfSession (System) 280, 281
 - dictionaries
 - Globals 152
 - Published 152
 - Direct I/O 316
 - directory, current, of child process 371
 - disableCommits: (UserProfile) 165
 - disaster recovery 355
 - disk drives
 - I/O among multiple extents 48
 - raw partitions 52
 - usage recommendations 43
 - disk failure 118
 - disk or repository full error 196
 - disk space
 - disk-full errors 196
 - distinguished name, and LDAP authentication 175
 - distributed system
 - and mid-level cache 94
 - DN (distinguished name), and LDAP 175
 - DUMP_OPTIONS 296
- ## E
- enableCommits: (UserProfile) 166
 - enableEpochGc (System) 266
 - enableSignaledAbortError 195
 - enableSignaledFinishTransactionError 196
 - environment variables 387–392
 - reserved names 392
 - epoch garbage collection 262, 266–271
 - benefits of 266
 - determining epoch length 267
 - forcing 266
 - limitations of 266
 - parameters for tuning 267
 - when 268
 - EpochGcCount (cache statistic) 272
 - #epochGcMaxThreads (GcUser parameter) 267
 - #epochGcPageBufferSize (GcUser parameter) 267
 - #epochGcPercentCpuActiveLimit (GcUser parameter) 267
 - #epochGcTimeLimit (GcUser parameter) 267
 - #epochGcTransLimit (GcUser parameter) 267
 - EpochNewObjs (cache statistic) 272
 - EpochPossibleDeadObjs (cache statistic) 272
 - EpochScannedObjs (cache statistic) 272
 - error messages
 - native language 380
 - errors
 - after restoring backup 223
 - disk full, diagnosing 196
 - extent already exists 106
 - extent already open 106
 - extent missing or access denied 106
 - fatal 55
 - in configuration files, option value 292
 - in configuration files, syntax 292
 - invalid password 174
 - key file 105

- object audit 134
 - shared page cache not attached 105
 - Stone response to Gem fatal error 55
 - stuck spin lock 119
 - transaction log missing 107
- ErrorSymbols 380
- examining
- user privileges 154, 161
- executable configuration files 291
- creating 288
 - defined 285
 - names 289
 - search for 287
 - setting permission 289
- expanding extents 197
- extent files 197
- see also *repository*
 - allocating space 294
 - allocation mode 47
 - checkpoint defined 185
 - creating new 49, 188
 - disk full condition 196
 - disk space, managing 197
 - free space in 186
 - group access to 81
 - location 45
 - moving to raw partition 53
 - naming 294
 - permissions for dynamically added 188
 - pre-growing 45, 295
 - reallocating objects in 190
 - removing 189
 - specifying size of 295
- F**
- failed login messages in log 174
- failover to standby system 241, 245
- false (kernel object) 377
- fastObjectAudit (Repository) 134
- file
- permissions for server files 80
- file descriptors 388
- FileControl (privilege) 150
- fileSize (Repository) 186
- fileSizeReport (Repository) 187
- findObjectsLargerThan:limit:(Object) 282
- findReferencePathToObject: (Repository) 282
- findReferences: (Object) 284
- findReferencesWithLimit: (Object) 284
- finishTransaction (signal) 196, 328
- forceEpochGc (System) 266
- fork-in-time scenario 233
- free frame cache
- specifying the size of 297
- free frame page server 60
- defined 25
 - log files 125
- free list page server
- benefits of 60
- free space
- in repository 186
- freeing repository space
- when 261
- freeSpace (Repository) 186
- full backup 211
- full logging
- how to manage 203
- fullBackupCompressedTo: (Repository) 218
- fullBackupCompressedTo:MBytes: (Repository) 218
- fullBackupTo: (Repository) 215
- G**
- garbage collection
- automatic mechanisms 249–271
 - backups and 217, 218
 - concepts 255
 - conflicts between mechanisms 263, 264
 - dead object, defined 257
 - determining epoch length 267
 - epoch collection 266–271
 - GcUser configuration parameters 262
 - identifying garbage 256
 - live object, defined 256
 - local object memory 256
 - markForCollection 264
 - mark/sweep, defined 260
 - object table sweep, defined 260
 - overview 260–261
 - pages, defined 256
 - parameters for tuning 267, 273–274, 280
 - possible dead objects, defined 260
 - reclaim 256, 263
 - resources reclaimed 256
 - shadow object, defined 257
 - transitive closure, defined 257
 - tuning reclaim 274
 - voting, defined 260
 - write set union sweep, defined 261
- GarbageCollection (privilege) 150

- GcGems
 - starting up all 276
 - tasks of 261
- GciStructsMd5 382
- GciStructsMd5String 380
- GciTsStructsMd5 382
- GciTsStructsMd5String 380
- GcLock 263
 - and markForCollection 264
 - backups and 217
- GcUser
 - changing parameters for 262
 - described 147
- GcVoteState, in monitoring reclaim 275
- Gem session process
 - configuration
 - file 65
 - run time access to 68
 - tuning 67
 - defined 64
 - linked and RPC 64
 - linked, setting up access 82
 - log files related to 127
 - remote from stone 89
 - starting 110
 - linked session 111
 - RPC session 112
 - troubleshooting 113
 - temporary object space, tuning 67
 - tuning configuration 67
- GEM_ABORT_MAX_CR5 296
- GemBuilder for C 291
- GemBuilder for Java 82
- GemBuilder for Smalltalk 291
- GEM_CACHE_WARMER_ARGS 296
- GEM_CACHE_WARMER_MID_CACHE_ARGS 297
- GEM_COMPRESS_TRANLOG_RECORDS 297
- gem.conf file 289
- gemConfigurationAt: (System) 68
- gemConfigurationReport (System) 68
- #GemConvertArrayBuilder 332
- #GemDropCommittedExportedObjs 332
- #GemExceptionSignalCapturesStack 332
- GEM_FREE_FRAME_CACHE_SIZE 297
- GEM_FREE_FRAME_LIMIT 297
- #GemFreeFrameLimit 298
- GemFreeFrameLimit 69
- GEM_FREE_PAGEIDS_CACHE 298
- #GemFreePageIdsCache 298
- GEM_HALT_ON_ERROR 298
- #GemHaltOnError 298
- GEM_KEEP_MIN_SOFTREFS 298
- #GemKeepMinSoftRefs 298
- GEM_KERBEROS_KEYTAB_FILE 299
- GEM_MAX_SMALLTALK_STACK_DEPTH 299
- GEM_NATIVE_CODE_ENABLED 299
- #GemNativeCodeEnabled 299
- gemnetobject** executable
 - mapping 290
 - RPC session and 113
- GEM_PG5VR_COMPRESS_PAGE_TRANSFERS 300
- #GemPgsvrCompressPageTransfers 300
- GEM_PG5VR_FREE_FRAME_CACHE_SIZE 101
- GEM_PG5VR_FREE_FRAME_CACHE_SIZE 300
- GEM_PG5VR_FREE_FRAME_LIMIT 300
- GEM_PG5VR_UPDATE_CACHE_ON_READ 101, 301
- #GemPgsvrUpdateCacheOnRead 301
- GEM_PG5VR_USE_SSL 301
- #GemPomGenPruneOnVote 306
- GEM_PRIVATE_PAGE_CACHE_KB 301
- GEM_READ_AUTH_ERR_STUBS 301
- #GemReadAuthErrStubs 301
- GEM_REPOSITORY_IN_MEMORY 302
- #GemRepositoryInMemory 302
- GEM_RPCGCI_TIMEOUT 302
- GEM_RPC_KEEPALIVE_INTERVAL 302
- gemsetup.csh 104, 109
- gemsetup.sh 104, 109
- GEM_SOFTREF_CLEANUP_PERCENT_MEM 303
- #GemSoftRefCleanupPercentMem 303
- GEM_STATMONITOR_ARGS 303
- GEM_STATMONITOR_MID_CACHE_ARGS 303
- GEMSTONE 111
 - description of 387
 - used by config files 40
- GemStone
 - component overview 63
 - configuration files used in 286
 - privileges required for system
 - administration tasks 145
 - privileges, defined 149
 - shutting down repository 116
 - avoid **kill -9** 117
 - starting repository monitor 104
 - SymbolDictionaries, used in symbol
 - resolution 151
 - user ID, defined 146
- GemStone login authentication 166
- GEMSTONE_ADMIN_GC_LOG_DIR
 - description of 387
- GemStoneError 380
- GEMSTONE_EXE_CONF 286, 287–288, 289, 387, 110
- GEMSTONE_GLOBAL_DIR 32, 387

- gemstone.hostid (host identifier) 32
 - GEMSTONE_KEEP_LOG 388
 - GEMSTONE_KEYRING_TABS 388
 - GemStone_Legacy_Streams 380
 - GEMSTONE_LIB 388
 - GEMSTONE_LOG 388
 - GEMSTONE_MAX_FD 388
 - GEMSTONE_NRS_ALL 74, 388, 371
 - GemStone_Portable_Streams 380
 - GEMSTONE_RECLAIM_GC_LOG_DIR 388
 - GEMSTONE_SPCMON_STARTUP_TIMELIMIT 388
 - GEMSTONE_SYMBOL_GEM_LOG_DIR 126, 388
 - GEMSTONE_SYS_CONF 285, 286–287, 288, 389
 - GEM_TEMPOBJ_AGGRESSIVE_STUBBING 303
 - GEM_TEMPOBJ_CACHE_SIZE 67, 250, 304
 - and multi-threaded operations 279
 - GEM_TEMPOBJ_CONSECUTIVE_MARKSWEEP_LI
MIT 304
 - GEM_TEMPOBJ_MESPACE_SIZE 251, 305
 - GEM_TEMPOBJ_OOMSTATS_CSV 254
 - GEM_TEMPOBJ_OOMSTATS_CSV 305
 - GEM_TEMPOBJ_OOPMAP_SIZE 251, 305
 - GEM_TEMPOBJ_POMGEN_PRUNE_ON_VOTE 306
 - GEM_TEMPOBJ_POMGEN_SCAVENGE_INTERVAL
306
 - #GemTempObjPomgenScavengeInterval 306
 - GEM_TEMPOBJ_POMGEN_SIZE 251, 306
 - GEM_TEMPOBJ_SCOPES_SIZE 306
 - global session statistics 142
 - Globals (system globals dictionary) 152
 - initial contents of 381
 - group:authorization:
 - (GsObjectSecurityPolicy) 160
 - groups
 - access to extents 81
 - adding a new user to 155
 - and AllGroups system object 379
 - and object security policy authorization 149
 - defined 154
 - removing from an object security policy's
authorization list 160
 - GS_CFTIME 389, 132
 - GS_CORE_TIME_OUT 389
 - GS_DEBUG_COMPILE_TRACE 253
 - GS_DEBUG_PAM 389
 - GS_DEBUG_SHARED_MEM 389
 - GS_DEBUG_SSL_LOG_DIR 389
 - GS_DEBUG_VMGC_MKSW_MEMORY_USED_SOFT_
BREAK 253, 389
 - GS_DEBUG_VMGC_MKSW_PRINT_C_STACK 253,
389
 - GS_DEBUG_VMGC_MKSW_PRINT_STACK 253,
389
 - GS_DEBUG_VMGC_PRINT_MKSW 389, 253
 - GS_DEBUG_VMGC_PRINT_MKSW_MEMORY 253,
390
 - GS_DEBUG_VMGC_PRINT_MKSW_MEMORY_USED
253, 390
 - GS_DEBUG_VMGC_PRINT_SCAV 390, 253
 - GS_DEBUG_VMGC_PRINT_TRANS 390
 - GS_DEBUG_VMGC_SCAV_PRINT_C_STACK 253,
390
 - GS_DEBUG_VMGC_SCAV_PRINT_STACK 253,
390
 - GS_DEBUG_VMGC_VERBOSE_OUTOFMEM 254,
390
 - GS_DEBUG_VMGC_VERIFY_MKSW 254, 390
 - GS_DEBUG_VMGC_VERIFY_SCAV 254, 390
 - GS_DEBUG_VM_PRINT_TRANS 253
 - GsDefaultIndexOptions 380, 382
 - GS_DISABLE_CHARACTER_TABLE_LOAD 390
 - GS_DISABLE_KEEPLIVE 390
 - GS_DISABLE_SHMDT 391
 - GS_DISABLE_SIGNAL_HANDLERS 391
 - GS_DISABLE_WARNING 390
 - GS_FORCE_CLEAN_LOG_FILE_DELETE 391
 - GS_KEEP_ALL_LOGS 391
 - gslist** command 108, 341
 - GsObjectInventory
and GcLock 263
 - and repository scan 136
 - GS_PAGE_MGR_PRINT_REMOTE_STACKS 391
 - GS_WRITE_CORE_FILE 391
- ## H
- high-availability systems, setting up standbys for
239
 - host identifier 32
 - hot standby 242–246, 350, 352
- ## I
- IcuLibraryVersion 382
 - identifying garbage 256
 - identifying large objects in the repository 281
 - INCLUDE 307
 - initialization
 - executing code on gem login 182
 - instance creation
 - and InstancesDisallowed 380
 - InstancesDisallowed 380
 - invalid password error 174
 - IPv6 318

K

keepalive, network option 88
 Kerberos 178–181, 299
 KEYFILE 307
 killing Gem or Stone processes 117

L

large objects, identifying in the repository 281
 large repositories
 special considerations 61
 lastLoginTime 146
 LDAP authentication 175
 Legal characters in a Stone name 354, 355
 licensing keyfile
 setting the location of 307
 linked vs. RPC application 64
 live object
 defined 256
 local object memory
 defined 256
 organization 249
 lock files 31
 log files 122–126
 for child processes 82
 for RPC Gems 82
 free frame page server 125
 Gem session process 127
 NetLDI 126, 128
 Page Manager 126
 Symbol Gem 126
 write access for 81
 login
 executing code on gem 182
 login authentication
 GemStone 166
 LDAP 175
 UNIX 174
 login object security policy, described 149
 login parameters 72
 loginHook
 182
 #LogOriginTime 332
 logOriginTime (Repository method) 205
 logreceiver 243, 244, 245, 349, 360
 logsender 242, 244, 245, 351, 361
 LOG_WARNINGS 307
 low memory
 signal 254
 LZ4 compression library 300, 324

M

manual transaction mode 121, 195, 217
 markForCollection
 conflicts with other garbage collection 264
 fast version 265
 scheduling with cron 265
 markForCollection
 conflicts with other garbage collection 263
 scheduling 265
 markForCollection (Repository) 262
 markForCollectionWithMaxThreads:...
 265
 marking garbage repository-wide 262
 marking objects for garbage collection 264
 mark/sweep
 defined 260
 master repository (in hot standby) 239, 242
 #maxTransactionDuration (GcUser
 parameter) 273
 mE
 region of temporary object memory 249
 memory
 and multi-threaded operations 279
 local object 249
 signalling on low 254
 temporary object 250
 memory space
 for multi-threaded operations
 calculating 279
 mid-level cache
 connection methods 95
 in distributed system 94
 reporting methods 96
 MinusInfinity (kernel object) 377
 MinusQuietNaN (kernel object) 377
 MinusSignalingNaN (kernel object) 377
 modes
 allocation 36
 debugging (NetLDI) 353
 full logging 25, 50, 199, 201
 guest (NetLDI) 353
 manual transaction 121
 partial logging 51, 201, 207
 transaction logging 49
 modifying
 another user's ID 158
 another user's password 158, 159
 monitoring
 cache statistics, programmatic access 138
 MtActiveThreads 279
 MtPercentCpuActiveLimit 278, 279
 MtThreadsLimit 278, 279

multi-threaded operations
 backup 217
 memory impact of 279
 monitoring 279
 session use and 217
 tuning 278

multi-threaded scan 278
 defined 278

N

name, stone legal name limits 354, 355
 Nameless (predefined system account) 147
 naming
 configuration file options 291
 executable configuration files 289
 extent files 294

Native Code 68
 NativeLanguage 380
 NetLDI
 debug mode 100
 default name 73
 defined 85
 in GEMSTONE_NRS_ALL 74
 list of 108
 log files 32, 126, 128
 permissions for executable netldid 78
 ports 73
 starting 109, 353
 troubleshooting 109
 well-known port 353

network
 disrupted communications 88
 Gem session process on Stone's machine 90
 keepalive option 88
 linked application on remote machine 98
 log files for spawned processes 82
 NetLDI 85
 setting up remote sessions 89
 shared page cache and 86
 troubleshooting remote logins 96
 typical configurations 83

network resource string
 #auth modifier 78

new
 region of temporary object memory 249

nil (kernel object) 377
 NoGsFileOnClient (privilege) 151
 NoGsFileOnServer (privilege) 151
 NoPerformOnServer (privilege) 150
 NotTranloggedGlobals 380
 NoUserAction (privilege) 151

NRS (network resource string) 28, 369
 GEMSTONE_NRS_ALL 371

O

object audit 134
 repairing errors 135

object memory
 organization of 249

object security policy
 changing a user's default 160
 changing the authorization of a 160
 unit of authorization 149
 used in read/write authorization 159

Object Server, see *Stone repository monitor*

object table
 loading at startup 61

object table sweep
 defined 260

objectAudit (Repository) 134
 objectAuditWithMaxThreads:percentCpu
 ActiveLimit: (Repository) 134

objects, large, identifying in the repository 281

ObjectSecurityPolicies
 predefined 378

ObjectSecurityPolicy
 predefined instances 378

ObjectSecurityPolicyCreation (privilege) 150
 ObjectSecurityPolicyProtection (privilege) 150
 #objsMovedPerCommitThreshold (GcUser
 parameter) 273

offline extent snapshot backup 211, 212

old
 region of temporary object memory 249

oldestLogFileIdForRecovery
 (Repository) 203

oldestLogFileIdForRecovery (Repository)
 204, 206

online extent snapshot backup 211

onlinebackup.sh
 file in GemStone examples directory 215

oopHighWater value
 and multi-threaded operations 279

/opt/gemstone/ used for GemStone files 31,
 81, 82

OtherPassword (privilege) 150
 OutOfMemory error 250

P

page audit 132
 Page Manager
 log files 126

- page server process
 - free frame page server 60
 - tasks of 59
 - pageaudit** command 344
 - pages
 - defined 256
 - PagesNeedReclaimSize, in monitoring reclaim 275
 - password
 - configuring authentication 166
 - constraining choices 167–173
 - modifying another user's 158, 159
 - modifying your own 158
 - percentCpuActive 279
 - perm
 - region of temporary object memory 249
 - permission, setting for executable configuration files 289
 - permissions
 - file 80
 - for shared page cache 310
 - PlusInfinity (kernel object) 377
 - PlusQuietNaN (kernel object) 377
 - PlusSignalingNaN (kernel object) 377
 - pom
 - region of temporary object memory 249
 - ports
 - NetLDI 73
 - shared page cache monitor 87
 - Stone 87
 - POSIX 354, 355
 - possible dead objects
 - defined 260
 - PossibleDeadObjs, in monitoring reclaim 275
 - pre-growing repository extents 45, 295
 - primary repository (in standby systems) 239
 - printing configuration options 296
 - privileges
 - adding to a user's 154, 161
 - assigning to a new user 155
 - defined 149
 - deleting a user's 161
 - examining a user's 154, 161
 - redefining a user's 154, 162
 - removing from user 161
 - required for system administration tasks 145
 - process slots
 - for cache statistics 139
 - profiling
 - repository contents 136
 - pstack** command 345
 - PublishedObjectSecurityPolicy 378
 - Publishers (predefined group) 153
 - purging unneeded objects 256
- ## R
- RAID devices 33, 45
 - raw partitions
 - changing to and from 53
 - removing old contents 346
 - setting up 52
 - use recommended 33, 43
 - read/write authorization 159
 - and object security policies 149
 - reclaim
 - garbage collection task 263
 - GcUser configuration parameters affecting 273
 - parameters for tuning 273–274
 - tuning 278
 - Reclaim Gem 261
 - and reclaim 272
 - configuring 273, 276
 - ReclaimCount, in monitoring reclaim 275
 - #reclaimDeadEnabled (GcUser parameter) 273
 - ReclaimedPagesCount, in monitoring reclaim 275
 - reclaiming system resources 256
 - #reclaimMinFreeSpaceMb (GcUser parameter) 273, 275
 - #reclaimMinPages (GcUser parameter) 273
 - recovery
 - after full disk error 196
 - after NetLDI startup failure 109
 - after Stone startup failure 104
 - after unexpected shutdown 117
 - redefining a user's privileges 154, 162
 - references to repository objects
 - searching for 284
 - remote logins
 - troubleshooting 96
 - remoteCachesReport (System) 96
 - removedbf** command 346
 - archiving transaction logs 204
 - removing
 - a user from the system 155
 - a user's privileges 161
 - extent 189
 - privilege from a user 161
 - symbolList from user 163
 - repair (Repository) 135
 - repository
 - see also *extent files* and *transaction logs*
 - audit at object level 134

- audit at page level 132
 - backups, see *backups*
 - checkpoint frequency 58
 - continuous restore mode 243
 - disaster recovery 355
 - disk full condition 196
 - error when below free space threshold 197
 - free space in 186
 - growth of 186, 255
 - identifying large objects in 282
 - marking garbage in 262
 - object references, searching for 284
 - oldest log needed for recovery 204
 - page fragmentation 194
 - profiling 136
 - running multiple 38
 - running warm or hot standby 239
 - shrinkage, when 261
 - shrinking to minimum size 191
 - shutting down 116
 - single instance of 378
 - standby systems 239
 - starting monitor 104
 - transaction logs, defined 25
 - updating views of extents 188
 - when free space appears in 272
- resolving symbols, `symbolList` used in 151
- restore mode 243
- `restoreFromBackup`: (Repository) 224
- `restoreStatus` (Repository) 220, 231
- `restoreStatusOldestFileId` (Repository) 206
- restoring the GemStone repository to a point in time 235
- RPC Gems
- configuration file name 289
- RPC vs. linked application 64
- ## S
- scan
- multi-threaded 278
- scheduling `markForCollection` 265
- search for
- executable configuration files 287
 - references to repository objects 284
 - system-wide configuration file 286
- secure mode, NetLDI 76
- security 166
- disabling inactive accounts 171
 - finding disabled accounts 164
 - last login by account 172
 - limiting concurrent sessions by user 173
 - login failures
 - disabling further 174
 - logging 174
 - passwords
 - aging 170
 - clearing disallowed list of 170
 - constraining choice of 167
 - disallowing certain 169
 - disallowing reuse of 169
 - login limit under a 173
 - warning of expiration 171
 - when last changed 171
 - see also *passwords*
 - `services.dat`
 - file in GemStone system directory 290
 - session statistics 142
 - `SessionAccess` (privilege) 150
 - `#SessionInBackup` 332
 - `SessionPriority` (privilege) 151
 - sessions
 - current session names 116
 - determining session holding up reclaim 281
 - find who is logged in 114
 - finding process id of 115, 280, 281
 - identifying current 281
 - setting
 - full transaction logging 329
 - permission, executable configuration files 289
 - `setuid` bit
 - for Gem session processes 81
 - on executable files 80
 - shadow object
 - contrasted with dead object 257–260
 - defined 257
 - shared memory
 - access by Gems 81
 - utility to check system 42
 - shared page cache
 - cleanup after **kill -9** 117
 - disconnect error 119
 - enabling 66
 - for remote Gem session processes 66
 - maximum processes 309
 - monitor process 26, 119
 - log file 124, 125, 126
 - on remote machine 91
 - size 310
 - spin lock attempts 58
 - stuck spin lock 119
 - timeout of remote 327

- shared system objects, in Globals dictionary 152
- shmdt() 391
- shrinking the repository 191
- SHR_NUM_FREE_FRAME_SERVERS 307
- SHR_PAGE_CACHE_LARGE_MEMORY_PAGE_POLICY 308
- SHR_PAGE_CACHE_LOCKED 309
- SHR_PAGE_CACHE_NUM_PROCS 66, 309
- SHR_PAGE_CACHE_NUM_SHARED_COUNTERS 309
- SHR_PAGE_CACHE_PERMISSIONS 310
- SHR_PAGE_CACHE_SIZE_KB 42, 66, 310
- shrpcmonitor 26
- SHR_SPIN_LOCK_COUNT 58, 310
- SHR_TARGET_FREE_FRAME_COUNT 311
- SHR_WELL_KNOWN_PORT_NUMBER 87, 311
- shutdown message 118, 119
- sigAbort 194, 195, 327
 - configuration parameter controlling 328
 - handler 195
- signal
 - sent on commit record backlog 194
 - sent on low memory 254
- Single sign-on 299
- slave repository (in hot standby) 239, 242
- #sleepTimeBetweenReclaimMs (GcUser parameter) 274
- #sleepTimeWithCrBacklogMs (GcUser parameter) 274
- Smalltalk
 - compiler, and symbol resolution 151
 - kernel classes, and Globals dictionary 152
 - methods, and GemStone privileges 149
- Smalltalk full backups 215
- space
 - determining usage in repository 136
- SpinLockCount 310
- standalone Gems 289
- standby systems 239
- startcachewarmer** command 347
- startCheckpointSync (System) 215
- starting a checkpoint 206
- starting GemStone 103
- startlogreceiver** command 243, 245, 349
- startlogsender** command 242, 245, 351
- startnetldi** command 353
- startNewLog (Repository) 206
- startstone** command 104, 355
 - when transaction logs are missing 108
- statistics
 - global 142
 - obtaining value by name 139
 - operating system 144
 - programmatic access to process CPU 143
 - see also under *cache statistics*
 - session 142
 - shared page cache 138
- statmonitor 138
- statmonitor** command 357
- STN_ADMIN_GC_SESSION_ENABLED 311
- #StnAdminGcSessionEnabled 311
- STN_ALLOCATE_HIGH_OOPS 311
- STN_ALLOW_NFS_EXTENTS 312
- STN_CACHE_WARMER 312
- STN_CACHE_WARMER_ARGS 312
- STN_CACHE_WARMER_SESSIONS 313
- STN_CACHE_WARMER_WAIT_MODE 313
- STN_CHECKPOINT_INTERVAL 59, 186, 313
- #StnCheckpointInterval 313
- STN_COMMIT_QUEUE_THRESHOLD 313
- #StnCommitQueueThreshold 313
- STN_COMMIT_RECORD_BM_CACHING 314
- STN_COMMIT_RECORD_QUEUE_SIZE 314
- STN_COMMITS_ASYNC 314
- STN_COMMIT_TOKEN_TIMEOUT 314
- STN_CR_BACKLOG_THRESHOLD 314, 194
- #StnCrBacklogThreshold 315
- #StnCurrentTranLogDirId 333
- #StnCurrentTranLogNames 333
- STN_DISABLE_LOGIN_FAILURE_LIMIT 174, 315
- #StnDisableLoginFailureLimit 315
- STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT 174, 315
- #StnDisableLoginFailureTimeLimit 315
- STN_DISKFULL_TERMINATION_INTERVAL 315
- #StnDiskFullTerminationInterval 315
- STN_EPOCH_GC_ENABLED 266, 316
- #StnEpochGcEnabled 316
- STN_EXTENT_IO_FLAGS 316
- STN_FREE_FRAME_CACHE_SIZE 316
- STN_FREE_SPACE_THRESHOLD 316
- #StnFreeSpaceThreshold 317
- STN_GEM_ABORT_TIMEOUT 317
- #StnGemAbortTimeout 317
- STN_GEM_LOSTOT_TIMEOUT 194, 317
- #StnGemLostOtTimeout 317
- STN_GEM_TIMEOUT 318
- #StnGemTimeout 318
- STN_GROUP_COMMITS 318
- STN_HALT_ON_FATAL_ERR 55, 56, 119, 318
- and Gem fatal errors 55
- #StnHaltOnFatalErr 56, 318
- STN_LISTENING_ADDRESSES 318
- #StnLogFileName 333

#StnLogGemErrors 333
 STN_LOGIN_LOG_DIR 319
 STN_LOGIN_LOG_ENABLED 319
 STN_LOGIN_LOG_HALT_ON_ERROR 320
 STN_LOGIN_LOG_MAX_SIZE 320
 #StnLoginsSuspended 333
 STN_LOG_IO_FLAGS 320
 STN_LOG_LOGIN_FAILURE_LIMIT 174, 321
 #StnLogLoginFailureLimit 321
 STN_LOG_LOGIN_FAILURE_TIME_LIMIT 174, 321
 #StnLogLoginFailureTimeLimit 321
 STN_LOOP_NO_WORK_THRESHOLD 321
 STN_MAX_AIO_RATE 322
 STN_MAX_AIO_REQUESTS 322
 STN_MAX_GC_RECLAIM_SESSIONS 322
 STN_MAX_LOGIN_LOCK_SPIN_COUNT 322
 STN_MAX_REMOTE_CACHES 323
 #StnMaxReposSize 333
 STN_MAX_SESSIONS 113, 217, 323
 #StnMaxSessions 333
 STN_MAX_VOTING_SESSIONS 323
 #StnMaxVotingSessions 323
 #StnMntMaxAioRate 322
 STN_NUM_AIO_WRITE_THREADS 323
 STN_NUM_GC_RECLAIM_SESSIONS 324
 #StnNumGcReclaimSessions 324
 STN_NUM_LOCAL_AIO_SERVERS 60, 324, 60
 STN_OBJ_LOCK_TIMEOUT 324
 #StnObjLockTimeout 324
 stnOopHighWater
 and multi-threaded operations 279
 #StnPageMgrCompressionEnabled 324
 STN_PAGE_MGR_MAX_WAIT_TIME 325
 #StnPageMgrMaxWaitTime 325
 StnPageMgrMaxWaitTime 57
 STN_PAGE_MGR_PRINT_TIMEOUT_THRESHOLD 325
 #StnPageMgrPrintTimeoutThreshold 325
 STN_PAGE_MGR_REMOVE_MAX_PAGES 325
 #StnPageMgrRemoveMaxPages 325
 STN_PAGE_MGR_REMOVE_MIN_PAGES 325
 #StnPageMgrRemoveMinPages 326
 STN_PGSRV_PORT_RANGE 326
 STN_PRIVATE_PAGE_CACHE_KB 326
 STN_REMOTE_CACHE_PGSRV_TIMEOUT 326
 #StnRemoteCachePgsvrTimeout 326
 STN_REMOTE_CACHE_STARTUP_TIMEOUT 326
 STN_REMOTE_CACHE_TIMEOUT 327
 #StnRemoteCacheTimeout 327
 #StnShrPcTargetPercentDirty 327
 STN_SHR_TARGET_PERCENT_DIRTY 327
 STN_SIGNAL_ABORT_AGGRESSIVE 327
 STN_SIGNAL_ABORT_CR_BACKLOG 327
 #StnSignalAbortCrBacklog 328
 STN_SMC_SPIN_LOCK_COUNT 328
 #StnSmcSpinLockCount 328
 #StnStandbyRole 333
 STN_STATMONITOR_ARGS 328
 STN_STONE_CACHE_STARTUP_TIMEOUT 328
 #StnSunsetDate 334
 STN_SYMBOL_GC_ENABLED 329
 #StnSymbolGcEnabled 329
 STN_SYMBOL_GEM_TEMPOBJ_CACHE_SIZE 329
 STN_TRAN_FULL_LOGGING 201, 203, 207, 329
 STN_TRAN_LOG_DEBUG_LEVEL 330
 #StnTranLogDebugLevel 330
 STN_TRAN_LOG_DIRECTORIES 52, 200, 201, 203, 205, 207, 330
 STN_TRAN_LOG_LIMIT 59, 201, 330
 #StnTranLogLimit 330
 #StnTranLogOriginTime 334
 STN_TRAN_LOG_PREFIX 200, 330
 STN_TRAN_LOG_SIZES 52, 200, 205, 207, 331
 STN_TRANQ_TO_RUNQ_THRESHOLD 331
 STN_WELL_KNOWN_PORT_NUMBER 87, 331
 Stone repository monitor
 AIO page servers 59
 configuration
 file 40
 run time access to 55
 defined 24
 extents 44
 on raw partitions 53
 file permissions for 80
 Gem fatal errors, response to 55
 identifying sessions logged in 114
 listing of 108
 raw partitions
 using 52
 recovery 117
 disk error 118
 disk full condition 196
 fatal error by Gem 119
 shared page cache error 119
 removing stale locks 341
 running multiple servers 38
 security, see *security*
 shared page cache 26
 tuning of 58
 starting 103
 troubleshooting 104
 status of 341
 swapping, excessive 27
 transaction logs 49
 on raw partitions 54

- stone.conf file 291
 - stoneConfigurationAt: (System) 55, 205
 - stoneConfigurationReport (System) 55, 56
 - stoplogreceiver** command 243, 245, 360
 - stoplogsender** command 242, 245, 361
 - stopnetldi** command 362
 - stopSession: (System) 116
 - delay for inactive sessions 116
 - stopstone** command 363
 - shutting down repository 116
 - StringConfiguration 380, 382
 - stuck spin lock error 119
 - Subscribers (predefined group) 154
 - suspending checkpoints 213
 - #sweepWsUnionMaxThreads (GcGem parameter) 280
 - #sweepWsUnionPageBufferSize (GcGem parameter) 280
 - #sweepWsUnionPercentCpuActiveLimit (GcGem parameter) 280
 - Symbol Gem
 - defined 25
 - log files 126
 - symbol list
 - adding to user 162
 - removing from user 163
 - symbol resolution 151
 - SymbolUser (predefined group) 154
 - syntax
 - configuration files 291
 - system
 - GemStone logs 123
 - objects, in Globals dictionary 152
 - shutdowns, diagnosing 117, 118, 196
 - System (predefined group) 153
 - system clock 33
 - SystemControl (privilege) 149
 - SystemObjectSecurityPolicy 378
 - SystemRepository 378
 - SystemUser
 - and AllUsers system object 379
 - and SystemObjectSecurityPolicy 378
 - described 146, 147
 - system-wide configuration files
 - defined 285
 - search for 286
- T**
- temporary object memory
 - error on low memory 250
 - temporary object space
 - finding out currently used 251
 - temporary objects 249
 - terminateSession:timeout: (System) 116
 - time, changes in system 33
 - Timeout, on SPC startup 388
 - Timestamp formatting 131
 - timeToRestoreTo: (Repository) 235
 - TimeZone 383–385
 - topaz 78, 99, 112
 - configuration files and 291
 - topaz** command 364
 - tranlogXXX.log (transaction log) 200
 - transaction logging
 - comparison of full, partial 201
 - enabling 49, 201, 329
 - partial logging checkpoint threshold 50, 201, 330
 - transaction logs
 - adding online 205
 - archiving 204
 - backups for 204
 - current log directory 205
 - current log file 205
 - current log size 205
 - disk full condition 208
 - disk space, managing 208
 - hot standby, use in 243
 - identifying checkpoints in 339
 - limiting size 240
 - log directories 200, 330
 - log not found error 107
 - log origin time 205
 - log size limit 200, 331
 - missing 236
 - moving to raw partition 54
 - oldest log needed for recovery 204
 - out of sequence 233
 - restarting stone without 108
 - restoring a subset of 232
 - restoring into backup 230–232
 - warm standby, use in 240
 - transaction mode, manual 121, 217
 - transaction record backlog 327
 - TransactionBacklog 194, 195, 327
 - transactionMode: (System) 195
 - transactions
 - checkpoint
 - starting 206
 - Transcript 380, 382
 - TranscriptStreamPortable 380, 382
 - transitive closure
 - defined 257
 - troubleshooting

NetLDI startup 109
 remote sessions 96
 session login 113
 Stone startup 104
 true (kernel object) 377
 tuning reclaim 278
 tz (TimeZone database) 384
 tzselect (TimeZone utility) 385

U

UNIX
 file system corruption 118
 UNIX authentication (for GemStone login) 174
 upgradeLogDir 391
 user groups
 and AllGroups system object 379
 assigning a new user to 155
 defined 154
 removing from an object security policy's
 authorization list 160
 used in object security policy authorization
 149
 UserPassword (privilege) 150
 UserProfile
 and AllUsers system object 379
 described 145
 userProfileForSession: (System) 116
 UserProfileGroup 153
 users
 current sessions 116
 default object security policy 146, 149
 dictionaries 151
 removing 163
 disabling inactive accounts 171
 finding disabled accounts 164
 group membership 146
 limiting concurrent sessions by same 173

listing all 158
 password 146, 148
 predefined users 146
 privileges 146, 149
 changing 161
 sessions holding up reclaim 280
 symbol list 151
 userId 148
 changing 158
 when last logged in 172
 why account disabled 165

UserSecurityData 146
 UTF-8, encoding in logs 122

V

verification of backups 218
 voting
 defined 260
 VSD 138, 141, 359

W

waitstone command 367
 warm standby 240–241
 warming, cache 296, 297, 312, 313, 347
 weighted allocation of extents 47
 write authorization 159
 write set union sweep
 defined 261
 tuning 280

Z

zdump (TimeZone utility) 385
 zic (TimeZone utility) 385
 zoneinfo (TimeZone database) 383